AGENT-BASED RESOURCE MANAGEMENT FOR GRID COMPUTING

Junwei Cao

A Thesis Submitted to the University of Warwick for
the Degree of Doctor of Philosophy

Department of Computer Science
University of Warwick
October 2001

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Prof. Graham R. Nudd, for offering me a great environment, plenty of opportunities, and freedom to be creative. I would like to thank my advisors, Dr. Darren J. Kerbyson and Dr. Stephen A. Jarvis, for giving me many detailed instructions on the ideas presented in this thesis.

I would also like to thank the members of High Performance Systems Group, for their help: Efstathios Papaefstathiou, John Harper, Stewart Perry, Daniel Spooner, James Turner, Ahmed Alkindi, Sirma Cekirdek, Dechao Wang, and Jiang Chen.

I would like to give special thanks to Prof. Malcolm McCrae and Dr. Li Xu. Without their efforts, I would not be able to get the opportunity to study at the University of Warwick.

Finally, I would like to thank my wife, Ms. Yu Han. Her love can always give inspirations for me to make progress on my work.

DECLARATION

This thesis is presented in accordance with the regulations for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree. The work described in this thesis has been undertaken by myself except where otherwise stated.

The various aspects concerning the PACE methodology have been published in [Cao2000]. Parts of the content in Chapters 4 – 7, concerning Sweep3D, A4, ARMS, and PMA, have been published in [Cao1999d], [Cao2000b], [Cao2001b] and [Cao2001] respectively. The detail introductions to the A4 methodology and the ARMS implementation have been accepted for journal publications in [Cao2001c] and [Cao2001d].

GLOSSARY

A4	Agile Architecture and Autonomous Agents
AARIA	Autonomous Agents at Rock Island Arsenal
ACL	Agent Communication Language
ACT	Agent Capability Table
ADEPT	Advanced Decision Environment for Process Tasks
API	Application Programming Interface
AppLeS	Application-Level Scheduler
ARMS	Agent-based Resource Management System (for Grid Computing)
ASCI	Accelerated Strategic Computing Initiative
AT	(PACE) Application Tools
C_ACT	Cached Agent Capability Table
CIMS	Computer Integrated Manufacturing System
CORBA	Common Object Request Broker Architecture
CORBA-LC	CORBA Lightweight Components
DHCP	Dynamic Host Configuration Protocol
DPM	Dynamic Performance Measurement
DPSS	Distributed-Parallel Storage System
EE	(PACE) Evaluation Engine
FTP	File Transfer Protocol

G_ACT	Global Agent Capability Table
GA	Genetic Algorithm
GRACE	Grid Architecture for Computational Economy
GUI	Graphical User Interface
GUSTO	Globus Ubiquitous Supercomputing Testbed
HAVi	Home Audio-Video interoperability
HLFD	Hierarchical Layered Framework Diagram
HMCL	Hardware Model Configuration Language
HTC	High Throughput Computing
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
JATLite	Java Agent Template, Lite
KQML	Knowledge Query and Manipulation Language
L_ACT	Local Agent Capability Table
LDAP	Lightweight Directory Access Protocol
MACIP	CIMS Application Integration Platform
MAS	Multi-Agent Systems
MDS	Metacomputing Directory Service
MPI	Message Passing Interface
OAS	Operational Administration System
OMG	Object Management Group
PACE	Performance Analysis and Characterisation Environment
PMA	Performance Monitor and Advisor
POEMS	Performance Oriented End-to-end Modelling System
PSE	Problem Solving Environment
PSL	Performance Specification Language
PVM	Parallel Virtual Machine
QoS	Quality of Service
RMI	Remote Method Invocation
RMS	Resource Management System
RSL	Resource Specification Language
RT	(PACE) Resource Tools
SDD	Self Device Describing
SDP	Service Discovery Protocol

SLA	Service Level Agreement
SLP	Service Location Protocol
SMP	Symmetric Multiprocessor
SNMP	Simple Network Management Protocol
SPE	Software Performance Engineering
SSDP	Simple Service Discovery Protocol
SUIF	Stanford University Intermediate Format
T_ACT	This Agent Capability Table
TCP/IP	Transmission Control Protocol/Internet Protocol
UPnP	Universal Plug and Play
XML	Extensible Makeup Language

ABSTRACT

A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capability. An ideal grid environment should provide access to the available resources in a seamless manner. Resource management is an important infrastructural component of a grid computing environment. The overall aim of resource management is to efficiently schedule applications that need to utilise the available resources in the grid environment. Such goals within the high performance community will rely on accurate performance prediction capabilities.

An existing toolkit, known as PACE (Performance Analysis and Characterisation Environment), is used to provide quantitative data concerning the performance of sophisticated applications running on high performance resources. In this thesis an ASCI (Accelerated Strategic Computing Initiative) kernel application, Sweep3D, is used to illustrate the PACE performance prediction capabilities. The validation results show that a reasonable accuracy can be obtained, cross-platform comparisons can be easily undertaken, and the process benefits from a rapid evaluation time. While extremely well-suited for managing a locally distributed multi-computer, the PACE functions do not map well onto a wide-area environment, where heterogeneity, multiple administrative domains, and communication irregularities dramatically complicate the job of resource management. Scalability and adaptability are two key challenges that must be addressed.

In this thesis, an A4 (Agile Architecture and Autonomous Agents) methodology is introduced for the development of large-scale distributed software systems with highly dynamic behaviours. An agent is considered to be both a service provider and a service requestor. Agents are organised into a hierarchy with service advertisement and discovery capabilities. There are four main performance metrics for an A4 system: service discovery speed, agent system efficiency, workload balancing, and discovery success rate.

Coupling the A4 methodology with PACE functions, results in an Agent-based Resource Management System (ARMS), which is implemented for grid computing. The PACE functions supply accurate performance information (e.g. execution time) as input to a local resource scheduler on the fly. At a meta-level, agents advertise their service information and cooperate with each other to discover available resources for grid-enabled applications. A Performance Monitor and Advisor (PMA) is also developed in ARMS to optimise the performance of the agent behaviours.

The PMA is capable of performance modelling and simulation about the agents in ARMS and can be used to improve overall system performance. The PMA can monitor agent behaviours in ARMS and reconfigure them with optimised strategies, which include the use of ACTs (Agent Capability Tables), limited service lifetime, limited scope for service advertisement and discovery, agent mobility and service distribution, etc.

The main contribution of this work is that it provides a methodology and prototype implementation of a grid Resource Management System (RMS). The system includes a number of original features that cannot be found in existing research solutions.

Chapter 1

INTRODUCTION

In fifty years, the raw speed of individual computers has increased by around one million times. However, they are still too slow for more and more scientific problems. For example, in some physics applications, data is produced by the fastest contemporary supercomputer, and it is clear that the analysis of this data would need much more computing power.

One solution to the computing power challenge leads to the research on Cluster Computing [Buyya1999]. Multiple individual computers can be linked into each other and work together to provide high computing capabilities. For example, the ASCI white system at Lawrence Livermore National Laboratory in the USA currently is the No. 1 supercomputer in the TOP500 list. This consists of SMP (Symmetric Multi-Processor) nodes, each containing 16 processors and clustered together using a high performance interconnect. Although clustering technologies enable a great deal of progress in providing computing power, a cluster remains a separate machine, dedicated to a specific purpose, and not being able to scale across organisation boundaries, which limits how large such a system can become.

With the rapid development of communication technologies, Internet Computing [Foster2000] provides another attempt towards supplying computing power in a more decentralised way. There are millions of powerful PCs around the world, however, most of them are idle much of the time. It is thought possible to harness these free CPU cycles so that scientists could solve important problems via the Internet. However, the real requirements may become much more complex. Email and the Web can only provide basic mechanisms for scientists to work together. Scientists may also want to link their data, their computers, and other resources together to provide a virtual laboratory [Foster2001]. The so-called Grid Computing technologies seek to make this possible.

1.1 Grid Computing

Civilisation has benefited from many successful infrastructures developed during 20th century. These include road systems, railways, the power grid, the telephone system, and the Internet. Once you press a light switch in a room, the light turns on. One can use it without knowing where the power comes from. The Internet is the latest important infrastructure, which is often referred to as the information highway. Given a domain name, you can get the information you want from your computer without knowing where the information comes from and how it reaches you.

The emerging concepts such as “The network is the computer”, “world-wide computer”, and “information power grid” [Leinberger1999] enable researchers in the high performance community to seek a new infrastructure that can provide not only information, but also high-end computing capabilities through networks. Once connected via your resource-short notebook to the network, it would be possible to run large scientific programs without worrying where the computing power comes from and whether it is a supercomputer in the US, Europe, or Japan that is actually doing computation for you.

A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capability [Foster1998]. It provides the protocols, services, and software development kits needed to enable flexible, controlled resource sharing on a large scale. The main components in the grid architecture include [Baker2001]:

- Grid Fabric - Comprising global resources geographically distributed and accessible from anywhere on the Internet. These resources might include computers (such as PCs and workstations running operating systems such as UNIX or Windows NT), supercomputers, clusters (running cluster operating systems or resource management systems), databases, and other special scientific instruments.
- Grid Services - Offering core services, such as information, communication, naming, resource management, performance analysis, visualisation, security and authentication, accounting, etc.
- Grid Tools - Providing high-level services allow programmers to develop grid applications. These services include languages, libraries, APIs, SDKs, debuggers, web tools, etc.
- Grid Applications - Grid-enabled applications developed using grid tools. There are many kinds of potential grid applications, such as wide-area distributed supercomputing, high-throughput computing, data-intensive computing, on-demand computing, etc.

The research into grid computing technologies can be split into three main phases:

- Exploration phase (- 1998). Several early attempts, which are now considered to be the classical projects in grid research, started with different motivation and together build an umbrella termed "Computational Grids". The key sign during this phase is the emergence of the GUSTO (Globus Ubiquitous Supercomputing Testbed), a prototype for future computational grids. Also the publication of the book in 1998, "The GRID – Blueprint for a New Computing Infrastructure" [Foster1998], indicate that the concept of the grid comes into being.

- Spreading phase (1998 - 2001). During this period, the concept of the grid has spread very rapidly. Researchers from the high performance community and others give annotations to the concepts from different views. Many projects begin to fit their research backgrounds into this new context. The key sign of this phase is that in March 2001, 360 researchers from USA, Europe, and Japan attended the first global grid forum (GGF1) held in Amsterdam (with 60 people having registration refused), and in May about 200 researchers from all over the world attended the first IEEE/ACM international symposium on cluster computing and the grid (CCGrid2001) held in Brisbane, Australia.
- Exploding phase (2001 -). Entering the new millennium, grid computing is considered to be an active research field with great potential and well known by most of computer scientists. Researchers from different fields of computer science will contribute work in this context. Companies support related activities on grid research. Governments begin to make plans to support native grid research and development. For example, the European Union gives 9.8 million euros funding over three years in support of the DataGrid project [Segal2000]. The UK Department of Industry also earmarked a large sum of money for their e-Science activities [Hey2001].

However, a practical grid environment does not yet exist. It is clear that the grid software infrastructure will be a large-scale distributed software system that is perhaps more complex than any existing software system. The most essential parts of the grid are its services, which act as middleware between grid resources and grid-enabled applications. Currently many grid-oriented software systems are being developed separately with different motivations, methodologies and tools. Many new ideas in them are important to accelerate the grid development. In order to integrate existing efforts and put the grid into practice, advanced software engineering methodology and technologies should be applied for the grid infrastructure development.

1.2 Software Agents

Software agents are becoming an important software development technology. The key sign of this trend is the emergence of diverse applications and approaches in many different areas [Bradshaw1997], including intelligent user interfaces [Lieberman1997], industry [Parunak1998], electronic commerce [Nwana1998], business process management [Jennings2000], digital libraries [Atkins1996], electronic meeting [Chen1996], entertainment [Maes1995], network management [Davison1998], and so on.

Agents are computer systems, capable of flexible, autonomous action in dynamic, unpredictable, typically multi-agent domains. Autonomy is the most essential feature, which differentiate the agent from other simple programs. Unfortunately, as mentioned in [Jennings1998], autonomy is a difficult concept to pin down precisely, but we mean it simply in the sense that the system should be able to act without the direct intervention of humans (or other agents), and should have control over its own actions and internal state.

There are basically two different ways for agents to achieve autonomy: intelligence and social ability. Intelligence means that an agent can achieve the autonomy by an AI approach within the ability of itself, such as personality, emotion, self-learning, life-like, knowledge reasoning, etc. Social ability means that an agent achieves its autonomy by relationships with the other agents in a Multi-Agent System (MAS), such as communication via an Agent Communication Language (ACL), coordination, negotiation, evolution, self-organisation, market mechanism, and mobility, etc.

For any new technology to be considered to be useful, it must offer either the ability to solve problems that have hitherto not been solved or the ability to solve problems that can already be solved in a significantly better (cheaper, more natural, easier, more efficient, or faster) way [Jennings2001b]. Software agents can be used to develop three classes of system:

- Open systems. An open system is one in which the structure of the system itself is capable of dynamically changing. The characteristics of such a system are that its components are not known in advance, can change over time, and may be highly heterogeneous. The best-known example of a highly open software environment is the Internet; and the grid is likely to also be an open system on a scale possibly larger than the Internet. The functionality is almost certain to require techniques based on negotiation or cooperation, which lie very firmly in the domain of MAS.
- Complex systems. The most powerful tools for handling complexity in software development are modularity and abstraction. Agents represent a powerful tool for making systems modular. They can provide a useful abstraction in just the same way those procedures, abstract data types, and objects provide abstractions. They allow a software developer to conceptualise a complex software system as a society of cooperating autonomous problem solvers. For many applications, this high-level view is simply more appropriate than the alternatives.
- Ubiquitous computing systems. Interaction between computer and user must become an equal partnership. The machine should not just act as a dumb receptor of task descriptions, but should cooperate with the user to achieve their goal. These considerations give rise to the idea of an agent acting as an expert assistant with respect to some application, knowledgeable about both the application itself and the user, and capable of acting with the user in order to achieve the user's goals.

Software agents have been accepted as a powerful high-level abstraction for the modelling of complex software systems like the grid software infrastructure. However, though in current grid-oriented software systems agent technology has been used in different ways, many new techniques developed in agent research have not yet been applied. The work described in this thesis integrates agent, performance, and scheduling technologies to implement one of the most important grid services, resource management.

1.3 Thesis Contributions

In this work the methodology, tools and implementation of agent-based resource management for grid computing are introduced. The performance prediction capabilities are used to provide quantitative data concerning the performance of sophisticated applications running on local high performance resources. At a metacomputing level, agents cooperate with each other and perform resource advertisement and discovery functions to schedule applications that need to utilise the available resources. The performance of the agent system can be also monitored, simulated, steered, and improved. The main contributions of this thesis include:

- Performance prediction driven QoS (Quality of Service) support of resource management and scheduling. Existing performance evaluation technologies can provide accurate prediction information regarding the execution of parallel and distributed applications. In this work, we integrate these performance prediction capabilities into resource management for grid computing. This is a key feature that differentiates this work from other solutions.
- Agent-based hierarchical model for service discovery. Agent hierarchies can be found in other agent applications [Ciancarini1999]. In this work, a hierarchy of homogenous agents with service advertisement and discovery capabilities is defined at a meta-level of a grid computing environment. This provides the first scalable agent-based resource management system for grid computing.
- Simulation-based performance optimisation and steering of agent-based service discovery. Most current grid resource management infrastructures focus on the implementation of data models and communication protocols. Performance issues have not been the key consideration of these systems. In our work, we focus more on performance optimisation of agent behaviours for service discovery. Several optimisation strategies and steering policies are provided and simulation tools and results are included

to show their impact on the overall system performance. To the authors' knowledge, this cannot be found in any of other works.

In summary, all of the above provide an available methodology and prototype implementation of agent-based resource management for grid computing, which can be used as a fundamental framework for further improvement and refinement. However, there are still some limitations on the system implementation aspect of this work.

- An existing PACE toolkit is used to provide performance prediction capabilities. For example, the PACE application performance modelling is based on the source code analysis, and a PACE resource model includes only static performance information of a resource.
- In the implementation of the agent-based grid resource management system, grid applications refer to only scientific computing applications that are computing intensive rather than data intensive, and grid resources are considered to be providers of high performance computing power rather than storage capabilities.
- While the performance monitoring and optimisation of agent behaviours are described as automatic processes, this is not implemented in the system described in this work. The use of the performance optimisation strategies and steering policies must be supervised by a system manager.

1.4 Thesis Outline

The introduction to agent-based resource management is divided into four parts. The PACE performance prediction capabilities are described using a parallel benchmarking program, Sweep3D. The agent hierarchy is included in the introduction of an A4 methodology. An initial implementation of the agent-based resource management for grid computing, ARMS, is given in an individual chapter. And the following chapter gives details of performance optimisation issues and an implementation of a performance monitor and advisor (PMA) in the ARMS.

The remaining parts of the thesis is organised in the following way:

Chapter 2 reviews existing performance techniques for parallel and distributed applications. The PACE toolkit, developed at the University of Warwick, is presented in greater detail. Several current solutions to grid resource management are also described and compared. Current challenges are then summarised.

Chapter 3 reviews existing agent infrastructure and service discovery techniques. The state of the art of agent technologies for grid computing is also summarised.

Chapter 4 introduces Sweep3D, a case study of performance evaluation using the PACE toolkit. The PACE performance model for Sweep3D is given in some detail. Validation results on different platforms with different data sizes are also included to show the prediction capabilities of PACE.

Chapter 5 presents the A4 methodology, agile architecture and autonomous agents, which can be used to build large-scale distributed systems that exhibit highly dynamic behaviour. The main issues include agent hierarchy, agent structure, agent capability tables, service advertisement and discovery, performance metrics. A simulator for A4 has been developed and is used to illustrate these issues.

Chapter 6 describes an implementation of an agent-based resource management system for grid computing, ARMS, which integrates PACE functions using the A4 methodology. The ARMS architecture and agent structure are presented in detail. The main modules in an ARMS agent include a communication module, an ACT (Agent Capability Tables) manager, a scheduler and a matchmaker. Experiments are also included to show the grid resource management capabilities of ARMS.

Chapter 7 discusses performance optimisation issues that arise from the agent system of ARMS. A special agent, PMA, acts as a performance monitor and advisor for ARMS, which is capable of performance modelling and simulation for agent resource discovery. Some optimisation strategies are suggested, including use of ACTs, limit resource lifetime, limit scope for resource advertisement and discovery, etc.

Chapter 8 draws conclusions from the work presented in this thesis and offers suggestions for future improvement to the methodology, tools, and implementation.

Chapter 2

Resource Management for Grid Computing

The grid, composed of distributed and often heterogeneous computing resources, is becoming the platform-of-choice for many performance-challenged applications. Proof-of-concept implementations have demonstrated that the grid environment has the potential to provide great performance benefits to parallel and distributed computing applications. The current research into grid computing aims to provide access to a multitude of processing systems in a seamless fashion. That is, from a user's perspective, applications may be executed on such a grid without the need to know which systems are being used, or where they are physically located. The overall aim of resource management is to efficiently schedule applications that need to utilise the available resources in the grid environment. Such goals within the high performance community will rely on accurate performance evaluation, analysis and scheduling.

2.1 Performance Evaluation

An increasing number of applications are being developed to run on parallel systems. An underlying goal in the use of high performance systems is to apply complex systems to achieve rapid application execution times. Whether there will be impressive gains in cost-performance make performance a key issue in parallel computing. For decades, the quantitative evaluation of computer performance has been applied to the entire life cycle of a system. These methods assist in the prediction, analysis, scheduling, and tuning of the performance of computers.

Numerous methodologies have been developed to evaluate the performance of computer systems. These can be organised into four main groups: benchmarking, analytical modelling, simulation, and monitoring. In benchmarking pre-defined workloads are run on systems to obtain performance measurements, which can be used as a basis for performance comparisons. Modelling methodologies require the construction of a mathematical or logical relationship that represents the behaviour of the system. The evaluation of this representation is performed by either analytical based techniques or by simulation. Monitoring tools can also be used to measure and analyse the performance of parallel systems. Performance studies often use more than one technique simultaneously to validate and verify the results of each other.

The techniques and tools that are being developed for the performance evaluation of parallel and distributed computing systems are many-fold, each having their own motivation and methodology. The main research projects currently in progress in this area are summarised in Table 2.1. A more detailed overview of previous performance evaluation methods and tools can be found in [Papaefstathiou1995b].

Name	Unit	Description
AppLeS [Berman1996]	Grid Computing Lab., Dept. of Computer Science and Engineering, Univ. of California, San Diego	This is an application-level scheduler using expected performance as an aid. Performance predictions are generated from structural models, consisting of components that represent the performance activities of the application.
CHAOS [Uysal1998]	High Performance Systems Software Lab., Dept. of Computer Science,	A part of this work is concerned with the performance prediction of large-scale data intensive applications on large-scale parallel machines. It includes a simulation-based framework to predict the performance of these applications on existing and

	Univ. of Maryland	future parallel machines.
PACE [Nudd2000]	High Performance Systems Lab., Dept. of Computer Science, Univ. of Warwick, UK	PACE is a performance prediction toolkit suitable for a non-performance expert. PACE supports the development of performance prediction models for sequential and parallel applications running on high performance systems. It is based on a layered characterisation methodology, and is an analytical approach that organises a performance model into three separate layers: application, parallelisation, and hardware.
Paradyn [Miller1995]	Paradyn Group, Dept. of Computer Science, Univ. of Wisconsin-Madison	Paradyn is a performance measurement tool for parallel and distributed programs. Paradyn scales to long running programs (hours or days) and large (thousand node) systems, and automates much of the search for performance bottlenecks. It can provide precise performance data down to the procedure and statement level. Paradyn is based on a dynamic notion of performance instrumentation and measurement. Unmodified executable files are placed into execution and then performance instrumentation is inserted into the application program and modified during execution.
Parsec [Bagrodia1998]	Parallel Computing Lab., Dept. of Computer Science, Univ. of California, Los Angeles	This is a parallel simulation environment for complex systems, which includes a C-based simulation language, a GUI (Pave), and a portable run-time system that implements the simulation operations.
POEMS [Deelman1998]	Parallel Computing Lab., Dept. of Computer Science, Univ. of California, Los Angeles, etc.	The aim of this work is to create a problem-solving environment for end-to-end performance modelling of complex parallel and distributed systems. This spans application software, run-time and operating system software, and hardware architecture. The project supports evaluation of component functionality through the use of analytical models and discrete-event simulation at multiple levels of detail. The analytical models include deterministic task graph analysis, and LogP, LoPC models [Frank1997].
GMA [Tierney2001]	Performance Working Group, Global Grid Forum	The goal of the development of a Grid Monitoring Architecture is to describe a common architecture with all the major components and their essential interactions in just enough detail that Grid Monitoring systems that follow the architecture can easily devise common APIs and wire protocols.

Table 2.1 Overview of Performance Evaluation Tools

The motivation behind the development of the Performance Analysis and Characterization Environment (PACE) at the University of Warwick was to provide quantitative data concerning the performance of sophisticated applications running on high performance systems [Cao2000]. The framework of PACE is a methodology based on a layered approach that separates software and hardware system components through the use of a parallelisation template. This is a modular approach that leads to readily reusable models, which can be interchanged for experimental analysis.

Each of the modules in PACE can be described at multiple levels of detail in a similar way to POEMS, thus providing a range of result accuracies but at varying costs in terms of prediction evaluation time. PACE is aimed to be used for pre-implementation analysis, such as design or code porting activities as well as for on-the-fly use in scheduling systems in similar manner to that of AppLeS. AppLeS is not originally motivated for grid computing but being improved to be utilised in a grid environment. In this work, PACE is integrated into an agent-based architecture to evaluate performance of grid applications. GMA is the only project that is developed in context of grid computing, however, it focuses more on performance monitoring than evaluation. The PACE methodology and toolkit are described in greater detail below.

2.2 PACE Methodology

The main concepts behind PACE include a layered framework and the use of associative objects as a basis for representing system components. An initial implementation of PACE supports performance modelling of parallel and distributed applications from object definition, through to model creation and result generation. These factors are described further below.

2.2.1 Layered Framework

Many existing techniques, particularly for the analysis of serial machines, use Software Performance Engineering (SPE) methodologies [Smith1990], to provide a representation of the whole system in terms of two modular components, namely a software execution model and a system model. However, for high performance computing systems, which involve concurrency and parallelism, the model must be enhanced. The PACE layered framework is an extension of SPE for the characterisation of parallel and distributed systems. It supports the development of three types of models: software model, parallelisation model and system (hardware) model. It allows the separation of the software and hardware model by the addition of the intermediate parallelisation model.

The framework and layers can be used to represent entire systems, including: the application, parallelisation and hardware aspects, as illustrated in Figure 2.1.

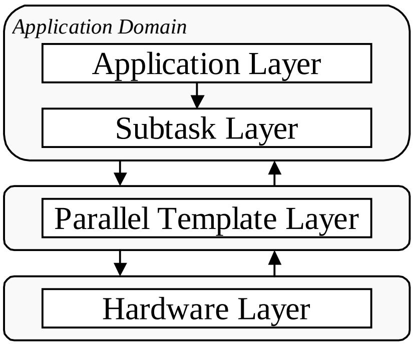


Figure 2.1 The PACE Layered Framework

The functions of the layers are:

- Application Layer – describes the application in terms of a sequence of parallel kernels or subtasks. It acts as the entry point to the performance study and includes an interface that can be used to modify parameters of a performance study.
- Application Subtask Layer – describes the sequential part of every subtask within an application that can be executed in parallel.
- Parallel Template Layer – describes the parallel characteristics of subtasks in terms of expected computation-communication interactions between processors.
- Hardware Layer – collects system specification parameters, micro-benchmark results, statistical models, analytical models, and heuristics to

characterise the communication and computation abilities of a particular system.

According to the layered framework, a performance model is built up from a number of separate objects. Each object is of one of the following types: application, subtask, parallel template, and hardware. A key feature of the object organisation is the independent representation of computation, parallelisation, and hardware. This is possible due to strict object interaction rules.

All objects have a similar structure, and a hierarchical set of objects representing the layers of the framework is built up into the complete performance model. An example of a complete performance model, represented by a Hierarchical Layered Framework Diagram (HLFD), is shown in Figure 4.2.

2.2.2 Object Definition

Each software object (application, subtask, or parallel template) is comprised of an internal structure, options, and an interface that can be used by other objects to modify its behaviour. A schematic representation of a software object is shown in Figure 2.2.

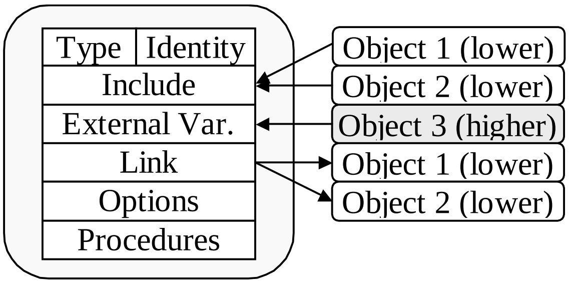


Figure 2.2 Software Object Structure

Each hardware object is subdivided into many smaller component hardware models, each describing the behaviour of individual parts of the hardware system. An example is shown in Figure 2.3 illustrating the main subdivision currently considered involving a distinction between computation, communication, memory [Harper1999] and I/O models.

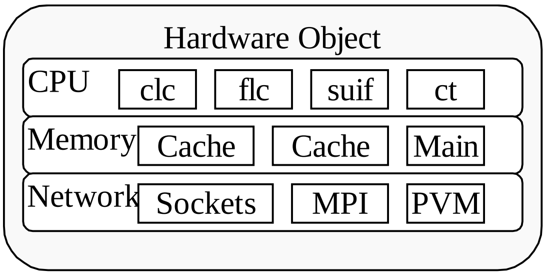


Figure 2.3 Hardware Object Structure

2.2.3 Model Creation

The creation of a software object in the PACE system is achieved through an application characterisation tool. It aids the conversion of sequential or parallel source code into a Performance Specification Language (PSL) [Papaefstathiou 1995] via the Stanford University Intermediate Format (SUIF) [Hall1996]. It performs a static analysis of the code to produce the control flow of the application, operation counts in terms of high-level language operations [Qin1991], and also the communication structure. This process is illustrated in Figure 2.4.

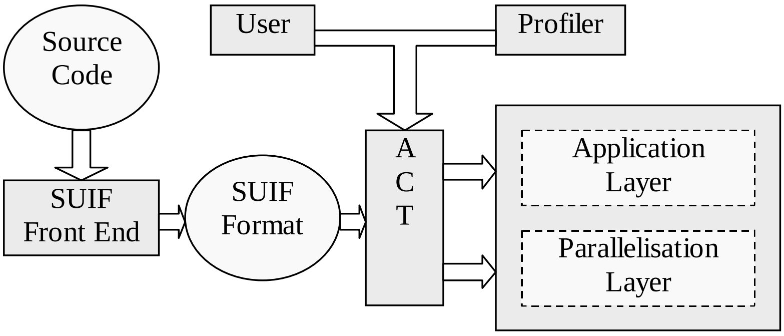


Figure 2.4 Model Creation Process

In PACE a Hardware Model Configuration Language (HMCL) allows users to create new hardware objects by specifying system-dependent parameters. On evaluation, the relevant sets of parameters are used, and supplied to the evaluation methods for each of the component models. An example HMCL fragment is illustrated in Figure 4.4.

2.2.4 Mapping Relations

There are strict mapping relations between source code of the application and its performance model. Figure 2.5 illustrates the way in which independent objects are abstracted directly from the source code and built up into a complete performance model, which can be used to produce performance prediction results.

The source code of the parallel application is firstly divided into several serial parts and an abstracted parallel part. Serial parts can be automatically converted into performance scripts using the PACE application characterisation tool. The parallel part can be converted into the corresponding parallel template line by line. The strict mapping relations make the model creation processes fast and straightforward. The user does not even need to understand the detailed parallelisation of the application.

The mapping relations are controlled by the PSL compiler and the PACE evaluation engine, which is described further in Chapter 4 through the use of the example application – Sweep3D.

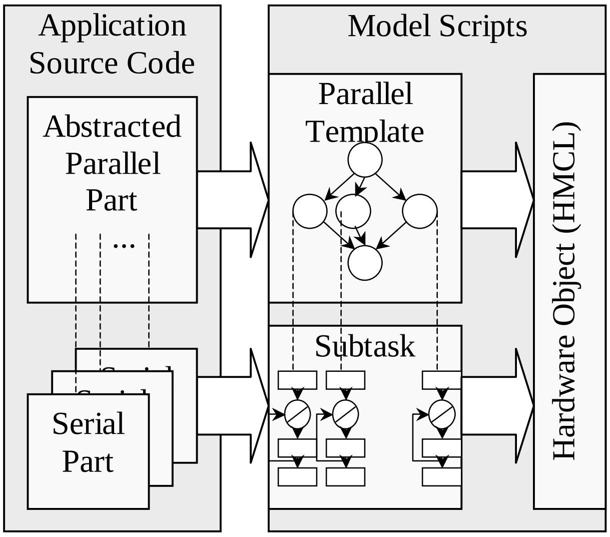


Figure 2.5 Mapping Relations

2.2.5 Evaluation Engine

Once all the necessary objects have been defined for a performance study, they can be combined and evaluated within the PACE evaluation engine. This involves the evaluation of the single application object, and all subtask objects, which in turn require the evaluation of associated parallel template objects and hardware objects. The sequence of steps performed during the evaluation of one subtask object is shown in Figure 2.6.

The *init* procedure of the subtask object is the entry point, which may call other procedures within the object. Parameters are linked to the currently active parallel template object (specified by the *option* command). The parallel template object is similarly evaluated and uses the hardware object. The result of the evaluation of the parallel template object is the execution time, which is returned to the application object. Further details can be found in [Papaefstathiou1998].

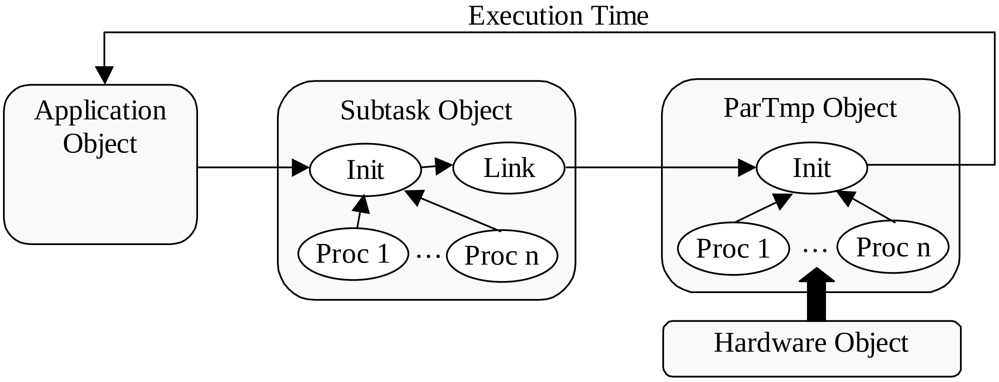


Figure 2.6 Evaluation Process of PACE Models

2.2.6 PACE Toolkit

The PACE methodology described above is implemented as a toolkit, which is summarised in Figure 2.7. The main components in the PACE toolkit include: application tools (AT), resource tools (RT), and an evaluation engine (EE).

- Application Tools: The Source Code Analyser can be used to convert sequential source code components into performance descriptions. Users can also edit these descriptions using the object editor or retrieve existing objects from a library. These performance descriptions are organised together into the PSL scripts of the application, which can be compiled into the application model. The application model is one of the inputs into the evaluation engine, which contains all of the application-level performance information.

- Resource Tools: The RTs provide several benchmarking programs to measure the performance of CPU, network interfaces (e.g. MPI and PVM), and memory aspects of hardware platforms respectively. The measurement results are represented in HMCL scripts and added to the system. The resource model is another input into the evaluation engine, which contains all of the system-level performance information.
- Evaluation Engine: The EE is the kernel part of the PACE toolkit, which executes completed performance models and produces evaluation results on time estimates, or trace information of the expected application behaviour. Important applications of prediction data include those of on-the-fly performance analysis [Kerbyson1998] and dynamic multi-processor scheduling [Perry2000], which can be applied for efficient resource management.

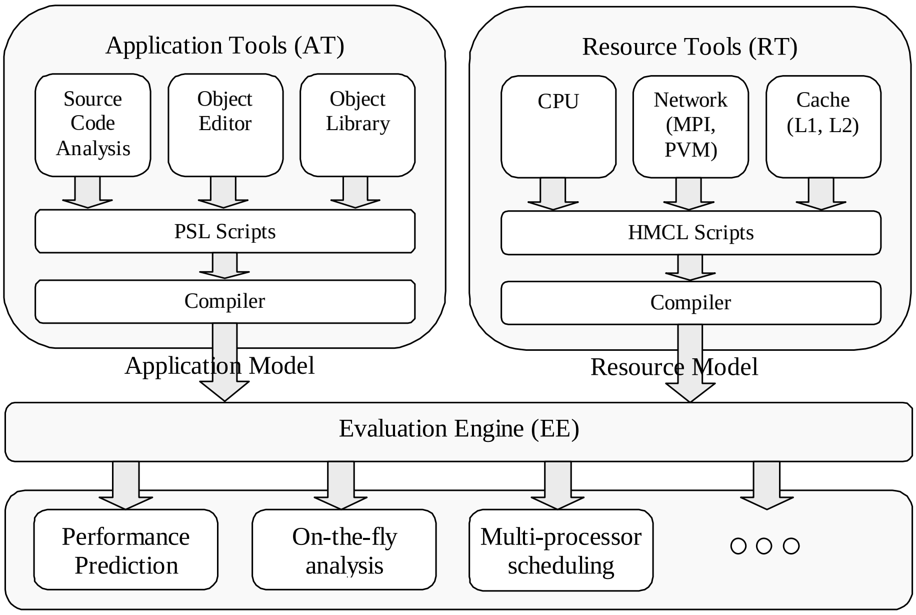


Figure 2.7 The PACE Toolkit

Some assumptions are made to simplify the PACE implementation. For example, the PACE application performance modelling is based on the source code analysis. The source code of the application is assumed to be available for performance modelling. A resource model in PACE can only include static performance information of a resource. The dynamic situation of the network traffic or CPU workload is not considered. The PACE toolkit is relatively smaller than many other performance evaluation tools, and some of its unique features (e.g. rapid evaluation time, reasonable accuracy, etc.) allow it to be applied to performance-driven resource management in a grid computing environment.

2.3 Grid Resource Management

The resource management is central to the operation of a grid. The basic function of resource management is to accept requests for resources from machines within the grid and assign specific machine resources to a request from the overall pool of grid resources for which the user has access permission. A resource management system matches requests to resources, schedules the matched resources, and executes the requests using the scheduled resources.

Several solutions have been offered that address to some extent the issues of resource management and scheduling for grid computing. Our work is different from these in a number of ways. Some of the principal existing grid projects and their resource management are described in Table 2.2. A good overview of grid resource management technologies can be found in [Krauter2000].

Name	Unit	Project Description	Resource Management
Condor [Litzkow1988] [Raman 1998]	Condor team, Dept. of Computer Science, Univ. of Wisconsin-Madison	The goal of the Condor project is to develop, implement, deploy and evaluate mechanisms and policies that support High Throughput Computing (HTC) on large collections of distributively owned computing resources.	Condor uses a classified advertisement (classad) matchmaking framework for flexible resource management in distributed environments with decentralised ownership of resources, which uses the matchmaker/entity (which can be both provider and requestor) structure. Features: extensible schema model; no QoS; network directory store; centralised queries discovery; periodic push advertisement.
DPSS	Data	The DPSS is a data block	DPSS uses a broker/agent

[Tierney 2000] [Brooks 1997]	Intensive Distributed Computing Group, Lawrence Berkeley National Laboratory	server, which provides high-performance data handling and architecture for building high-performance storage systems from low-cost commodity hardware components. This technology has been quite successful in providing an economical, high-performance, widely distributed, and highly scalable architecture for caching large amounts of data that can potentially be used by many different users.	architecture: agents are processes that monitors the state of the system; broker agent (or broker) is an agent that manages the information, filters information for clients, or performs some action on behalf of a client. Agents model their environment using an extensible set of Facts and act on their environment using a set of Tasks. Features: object model; no QoS; agent-based store; centralised queries discovery; periodic push advertisement.
Globus [Foster1997] [Czajkowski1998]	Mathematics and Computer Science Division, Argonne National Laboratory	The Globus system is intended to achieve a vertically integrated treatment of application, middleware, and network. A low-level toolkit provides basic mechanisms such as communication, authentication, network information, and data access. These mechanisms are used to construct various higher-level metacomputing services, such as parallel programming tools and schedulers. The long-term goal is to build a grid infrastructure, an integrated set of higher-level services that enable applications to adapt to heterogeneous and dynamically changing metacomputing environments.	The architecture distributes the resource management problem among distinct local manager, resource broker, and resource co-allocator components, and defines an extensible resource specification language (RSL) to exchange information about requirements. The information service within the architecture uses a Metacomputing Directory Service (MDS) [Fitzgerald1997], which adopts the data representations and API defined by the LDAP service [Yeong1995]. Features: extensible schema model; soft QoS; network directory store; distributed queries discovery; periodic push advertisement.
GRACE [Buyya2000]	School of Computer Science and Software Engineerin	GRACE (Grid Architecture for Computational Economy) is a new framework that uses economic theories in grid resource management and	Nimrod/G is a grid resource broker that allows managing and steering task farming applications (parameter studies) on computational grids. It follows an economic

	g, Monash University, Australia	scheduling. The components that make up GRACE include global scheduler (broker), bid-manager, directory server, and bid-server working closely with grid middleware and fabrics. The GRACE infrastructure also offers generic interfaces (APIs) that the grid tools and applications programmers can use to develop software supporting the computational economy.	(computational market) model for resource management and scheduling. It allows the study of the behaviour of output variables against a range of different input scenarios. Features: extensible schema model; hard QoS; relational resource info store; distributed queries discovery; periodic push/pull advertisement.
Legion [Grimshaw1999] [Chapin1999]	Dept. of Computer Science, Univ. of Virginia	Legion is an object-oriented metacomputing environment, intended to connect many millions of hosts ranging from PCs to massively parallel supercomputers. It manages billions of objects and allows users to write and run applications in an easy-to-use, transparent fashion. It unites machines from thousands of administrative domains into a single coherent system.	Legion uses a resource management infrastructure. The philosophy of scheduling is that it is a negotiation of service between autonomous agents, one acting on the part of the application (consumer) and one on behalf of the resource or system (provider). The components of the model are the basic resources (hosts and vaults), the information database, the schedule implementer, and an execution monitor. Features: extensible object model; soft QoS; object model store; distributed queries discovery; periodic pull advertisement.
NetSolve [Casanova1998]	Dept. of Computer Science, Univ. of Tennessee	NetSolve is a client-server system that enables users to solve complex scientific problem remotely. The system allows users to access both hardware and software computational resources distributed across a network. NetSolve searches for computational resources on a network, chooses the best one available, and using retry	The NetSolve agent operates both as a database and as a resource broker. The agent keeps track of information about all the servers in its resource pool, including their availability, load, network accessibility, and the range of computational tasks that they can perform. The agent then selects a server to perform the task, and the server responds to the client's request.

		for fault-tolerance solves a problem, and returns the answers to the user.	Features: extensible schema model; soft QoS; distributed queries discovery; periodic push advertisement.
Ninf [Sato1998] [Nakada 1998]	Computer Science Division, Electrotechnical Laboratory, Japan	Ninf is an ongoing global network-wide computing infrastructure project which allows users to access computational resources including hardware, software and scientific data distributed across a wide area network with an easy-to-use interface. Ninf is intended not only to exploit high performance in network parallel computing, but also to provide high quality numerical computation services and accesses to scientific databases published by other researchers. Computational resources are shared as Ninf remote libraries executable at a remote Ninf server.	In order to facilitate location transparency and network-wide parallelism, the Ninf metaserver maintains global resource information regarding computational server and databases, allocating and scheduling coarse-grained computation to achieve good global load balancing. The Ninf metaserver is a JAVA agent, a set of which gathers network information regarding the Ninf servers, and also helps the client to choose an appropriate Ninf server, either automatically or semi-automatically. Features: fixed schema model; no QoS; centralised queries discovery; periodic push advertisement.

Table 2.2 Overview of Grid Projects and their Resource Management

Grid resources are the entities such as processors or hosts that are managed by the resource management system. A local resource in the grid is usually a multi-processor or a cluster of machines, which are distributed geographically in a small scope, connected with high-speed networks, and administrated within the same organisation. These local resources may be far away from each other, connected via the Internet with irregular communications, and cross administrative domains. All these resources compose a global metacomputing environment, such as that illustrated in Figure 2.8.

The grid resource management functions are performed at both a meta and a local level. Each local high performance resource is managed by a local resource manager. A mechanism is also needed at a meta-level to coordinate the behaviours of multiple local resource managers so as to achieve high performance in the overall grid system.

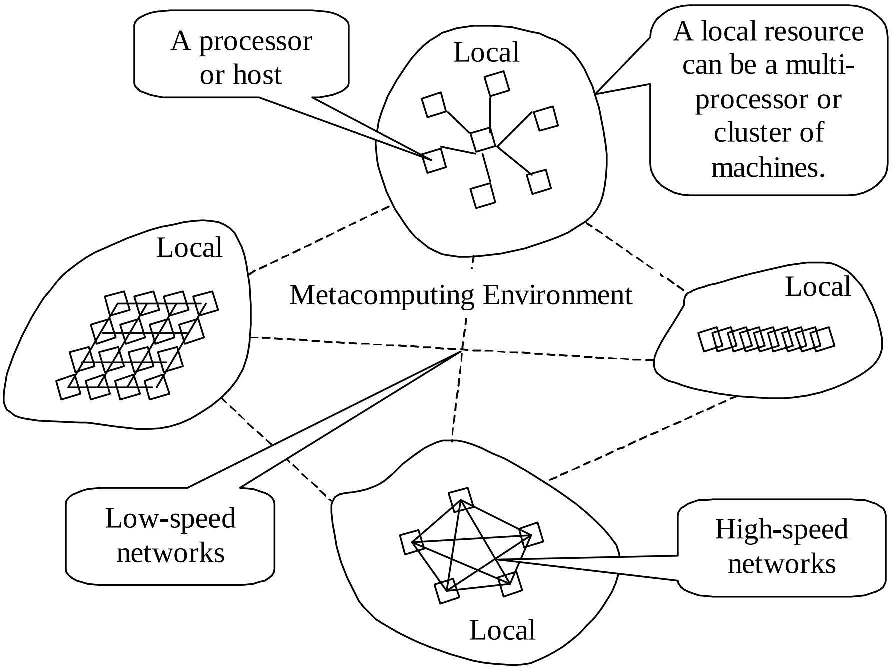

A processor or host
Local
A local resource can be a multi-processor or cluster of machines.
Local
Metacomputing Environment
Local
Low-speed networks
High-speed networks
Local

Figure 2.8 Grid Resources

The basic issues relating to metacomputing resource management include data representation and management, communication protocols, resource discovery and quality of service (QoS) support. The main issues related to local resource management are multi-processor scheduling, resource allocation and monitoring. These are introduced in detail below.

2.3.1 Data Management

The main data used in a resource management system is that used to describe the attributes and operations of a resource. Data management related issues include data representation and data storage.

A grid resource can be described by a corresponding resource model. The resource model determines how to describe and manage the grid resource. There are two basic approaches for data representation.

- Schema based approach. The data that comprises a resource is described in a description language along with some integrity constraints. The schema languages are further characterised by the ability to extend the schemas. In a fixed schema all elements of resource description are defined and cannot be extended. In an extensible scheme new schema types for resource descriptions can be added. Predefined attribute-value based resource models are in the fixed schema category. The resource specification language (RSL) used in Globus resource management is an extensible schema model. The Condor ClassAd approach using semi-structured data approach is also in the extensible schema category.
- Object based approach. In an object model scheme the operations on the resources are defined as part of the resource model. The object model can be predetermined and fixed as part of the definition of the resource management system. Also the resource model can provide a mechanism to extend the definition of the object model managed by the system. Legion uses extensible object models to describe resources in the system.

The resource information should be stored in the resource management system in proper organisation. It helps characterise the overall performance of the resource management system and determine the cost of implementing the resource management protocols since a resource discovery capability may be provided by the data storage implementation. There are two basic approaches to the storage of resource information in the system.

- Network directories. Network directory data storage is based on IETF standards, such as LDAP [Yeong1995] and SNMP [Case1988], or utilise specialised distributed database implementation. The information service in Globus resource management system uses a Metacomputing Directory Service (MDS), which adopts the data representations and API defined by the LDAP service.
- Distributed objects. This data storage approach utilise persistent object services that can be provided by a language-based model such as that

provided by persistent Java object implementations. Legion uses object model data storage.

The important difference between the distributed object and network directory approaches is that in network directories the schema and operations are separated with the operations defined externally to the data store schema. In an object oriented approach the schema defines the data and the operations.

The applications submitted from the grid users should also be attached to a corresponding application model, including the information on the requirements from the user on the application execution. The representation, storage, and transference of these application models are also very important and have impacts on the overall performance of the resource management system. Most of above issues on resource models can also be applied to application models, which will not therefore be discussed again in detail here.

2.3.2 Communication Protocols

Communication is a central issue for building distributed software systems. In a grid resource management system, different local resource managers need to communicate with each other to perform meta-level resource management functions. Communication protocols are needed as the basis of communication implementation. The implementation of communication enables different entities in a distributed system to communicate with each other. However, some common protocols are needed for them to understand each other.

Communication can be implemented by low-level Internet protocols, such as TCP/IP, FTP, and HTTP. The communication protocols can be pre-defined in the system using simple data structures. Many existing enterprise distributed system infrastructures, languages and platforms can also provide powerful support for data representation and communication. In the work described in this thesis, data representation and communication protocols have not been the key consideration. The resource management system focuses more on resource discovery, QoS support, and related performance issues.

2.3.3 Resource Advertisement and Discovery

A major function of a grid resource management system is to provide a mechanism for resources in the grid to be discovered and utilised by grid applications. Resource advertisement and discovery provide complementary functions. Discovery is initiated by a grid application to find suitable resources within the grid. Advertisement is initiated by a resource trying to find a suitable application that can utilise it. The overhead of matching resources and applications determines the efficiency of the system and determines the maximum resource utilisation that a resource management system can achieve in the grid computing environment. There are two approaches to resource advertisement and discovery in a grid resource management system: query based and agent based.

- Query-based: Network directory based mechanisms such as Globus MDS use parameterised queries that are sent across the network to the nearest directory, which then uses a query engine to execute the query against the database contents. Query based systems are further characterized depending on whether the query is executed against a distributed database or a centralized database. Legion also uses distributed query-based resource discovery, while centralised query-based resource discovery is adopted in most current computational grid projects, such as Condor, DPSS, NetSolve and Ninf.
- Agent-based: Agent based approaches send active code fragments across machines in the grid that are interpreted locally on each machine. Agents can also passively monitor and either periodically distribute resource information or respond to another agent. Thus agents can mimic a query based resource discovery scheme. Currently agent-based approaches can only be found in some service discovery projects (which will also be discussed in detail in the next chapter), such as 2K [Kon2000] and Bond [Boloni1999]. The agent-based resource management system described in this work aims to apply agent technologies in resource management for computational grids.

The major difference between a query based approach and an agent based approach is that agent based systems allow the agent to control the query process and make resource discovery decisions based on its own internal logic rather than rely on a fixed function query engine. Agent based resource discovery is inherently distributed.

2.3.4 QoS Support

In metacomputing resource management, resources should be discovered and corresponding resource information should be returned to the grid user according to QoS principles. As also described in [Krauter2000], our notion of QoS is not limited to network bandwidth but extends to the processing capabilities of the resources in the grid. Thus we focus on the degree that a grid can provide end-to-end QoS across all components rather than QoS only on the network.

There are two parts to QoS, admission control and policing. Admission control determines if the requested level of service can be given and policing ensures that the application does not violate its service level agreement (SLA). A resource management system that does not allow applications to specify QoS requirements in resource requests does not support QoS. Otherwise the QoS support can be classified into soft and hard support.

- Soft QoS support. An RMS that provides explicit QoS attributes for resource requests but cannot enforce service levels via policing provides soft QoS support. Most current grid systems (e.g. Globus, Legion, and NetSolve) provide soft QoS since most non real-time operating systems do not allow the specification of service levels for running applications and thus cannot enforce non-network QoS guarantees.
- Hard QoS support is provided when all nodes in the grid can police the service levels guaranteed by the resource management system. Nimrod/G in GRACE supports hard QoS through computational economy services of GRACE infrastructure.

The resource management system described in this thesis can also provide hard QoS support. The users need to define their requirements explicitly when they submit a resource request, which is similar to Nimrod/G. Unlike Nimrod/G, in which the grid resource estimation is performed through heuristics and historical information (load profiling), the performance prediction capability of grid resources is achieved via integrating PACE functions into the system.

There are some other functions that can be provided in meta-level grid resource management. For example, co-allocation problems arise when applications have resource requirements that can be satisfied only by using resources simultaneously at several sites. As described in [Foster1999], Globus resource management supports resource co-allocation, which, however, is not the key consideration in our implementation. In the sections below, brief introductions are given to two important issues related to local resource management.

2.3.5 Resource Scheduling

The scheduling on a local grid resource is a “multiple applications on multiple processors” problem. Applications arrive at the resource at different times with different requirements. Resource scheduling in a local resource manager is responsible for deciding when to start running an application, and how many processors should be dispatched to an application. There are two kinds of scheduling policies and corresponding metrics.

- Resource-oriented - maximising the utilisation of the resource. In a previous work done at Warwick [Perry1999], scheduling a number of parallel applications on a homogenous multi-processor machine is studied. It is achieved through just-in-time performance prediction (provided by PACE) coupled with iterative heuristic algorithms for optimisation of the utilisation of the resource.
- Application-oriented - meeting requirements from the applications. In the system described in this work, each application submitted from a grid user should be attached with explicit performance requirements. Local resource

scheduling focuses on meeting these requirements from the user point of view.

These two aspects of scheduling are related, but sometimes may conflict. There must be a balance in order to achieve both resource-oriented and application-oriented optimisation. Rescheduling is also a part of resource scheduling problem. The rescheduling characteristic of a resource management system determines when the current schedule is re-examined and the application executions reordered. There are two rescheduling approaches.

- Periodic or batch rescheduling approaches group resource requests and system events and process them at intervals. This interval may be periodic or may be triggered by certain system events. The key point is that rescheduling is done in batches instead of individual requests or events.
- Event driven online rescheduling performs rescheduling as soon the resource management system receives the resource request or system event.

The local resource scheduling is not the main focus of the work described in this thesis. However, in the following chapters, the related problems will be mentioned and discussed. An algorithm will also be given for an initial implementation to be used by meta-level resource management.

2.3.6 Resource Allocation and Monitoring

After applications are scheduled on a grid resource, resource allocation is responsible for running the application and returning the results. The local resource manger should be wrapped with parallel application execution environments like MPI and PVM. When the application begins running, the resource should be monitored and corresponding information can be used by local-level rescheduling or meta-level resource discovery. These will not be discussed in detail here.

2.4 Summary

A grid infrastructure is a large-scale distributed system with highly dynamic behaviours. This chapter introduces the research background to performance evaluation techniques and grid resource management issues. Previous work on the PACE toolkit at Warwick has been described in detail. In summary, the development of computational grids introduces two key challenges:

- Scalability: The grid may potentially encompass all high performance computing resources. A given component of the grid will have it's own functions, resources and environment. These are not necessarily geared to work together in the overall grid. They may be physically located in different organisations and may not be aware of each other.
- Adaptability: A grid is a dynamic environment where the location, type, and performance of the components are constantly changing. For example, a component resource may be added to, or removed from, the grid at any time. These resources may not be entirely dedicated to the grid; hence their computational capabilities will vary over time.

New software development technologies are needed for the implementation of the grid software infrastructure. Several new grid projects are utilising existing distributed computing technologies, such as CORBA (Common Object Request Broker Architecture) [Slama1999] and Jini [Amold1999].

CORBA is OMG's (Object Management Groups) open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. CORBA was not originally designed for developing high performance computing applications. Some work provides CORBA based tools that enable to use CORBA in different contexts. For example, in the work described in [Denis2001], portable parallel CORBA objects are provided as a new programming approach for grid computing, which can interconnect two MPI codes by CORBA without modifying MPI or CORBA. The work described in [Sevilla2001] makes use of the CORBA-LC (CORBA Lightweight Components) to provide a new network-centred reflective component model, which allows building distributed applications assembling binary independent components spread on the network. However, as mentioned in [Foster2001], such technologies only enable resource sharing within a single organization, and can not be used to address the concerns and requirements listed above.

A Jini system is a distributed system federating groups of users and resources, which is based on the Java platform. The work described in [Furmento2001] is a computational community that supports the federation of resources from different organisations, designed and implemented in Java and Jini. The service discovery technique in Jini is introduced in the next chapter.

Agent technologies have been used for the development of distributed software systems for several years. Multi-agent systems provide a clear high-level abstraction and a more flexible implementation of distributed infrastructures and applications. Multi-agent systems coupled with service discovery approaches are introduced in the following chapter.

Chapter 3

SERVICE DISCOVERY IN MULTI-AGENT SYSTEMS

The software infrastructure of the grid is an open, complex software system. Multi-agent technology is one of the ways to overcome the challenges in the development of the grid. Service has been accepted as the most important concept in this distributed system development, and service discovery is therefore considered an essential part in many distributed system infrastructures. In this chapter, we introduce in detail background research on service discovery in multi-agent systems, the technique of which will be used in our grid resource management system.

3.1 Multi-Agent Systems

Agent technologies have been developed for over ten years. Numerous theories, languages, tools, and applications have emerged in different fields [Cao1998]. Giving a short survey of multi-agent systems is a difficult task. However, there is an easy and direct way to obtain an impression on what a multi-agent system is by looking into several representative and successful multi-agent projects. Table 3.1 gives a list of 6 agent projects, including 3 multi-agent applications, 1 mobile agent project, 1 agent development tool and 1 agent communication language.

Name	Unit	Description
AARIA [Parunak 2001]	Michigan Manufacturing Technology Centre, etc.	AARIA is an industrial-strength agent-based factory scheduling and simulation system. Three persistent agents are Parts, Resources, and Unit Process. Interactions among these three persistent agents are modelled as transient agents, such as Engagements, Materials, Products, and Operations. Each transient agent has a six-phase life cycle: Inquiring, Committing, Committed, Available, Active, and Achieved.
ADEPT [Jennings2000b]	DAI Research Unit, Queen Mary and Westfield College, Univ. of London, UK	A business process is composed of a number of primitive functional activities or tasks. In any reasonably complex process, dependencies exist between the tasks and so they have to be executed in a controlled and ordered way. This execution invariably involves the consumption of resources. In most organisations, these resources are grouped into business units that control the way in which they are deployed. Within ADEPT, these business units are represented by autonomous software agents. The agents communicate with one another over a network and negotiate over how they can collaborate to manage the overall business process. To be consistent with the service-oriented philosophy, negotiation and collaboration are at the level of the services that agents offer to one another. In this case, a service is a packaging of tasks and other (sub-) services that allows an agent to offer or to receive from another agent some functional operation. A service can be reused as a component of another service and agents can take the role of provider (server) or customer (client) for services.
D'Agents [Brewington1999]	Dept. of Computer Science, Dartmouth College	A mobile agent is an executing program that can migrate during execution from machine to machine in a heterogeneous network. On each machine, the agent interacts with stationary service agents and other resources to accomplish its task. Mobile agents are particularly attractive in distributed information-retrieval applications. By moving to the location of an information resource, the agent can search the resource locally, eliminating the transfer of intermediate results across the network and reducing end-to-end latency.
JATLite [Jeon2000]	Agent Based Engineering Group, Centre for Design Research, Stanford Univ.	JATLite (Java Agent Template, Lite) is a package of programs written in the Java language that allow users to quickly create new software agents that communicate robustly over the Internet. JATLite provides a basic infrastructure in which agents register with an Agent Message Router facilitator using a name and password, connect/disconnect from the Internet, send and receive messages, transfer files, and invoke other programs or actions on the various computers where they are running.

		JATLite especially facilitates the construction of agents that send and receive messages using the emerging standard communications language, KQML.
KQML [Labrou1999]	Laboratory for Advanced Information Technology, Computer Science and Electrical Engineering, University of Maryland, Baltimore County	KQML, the Knowledge Query and Manipulation Language, is a language and protocol for exchanging information and knowledge. KQML is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. KQML can be used as a language for an application program to interact with an intelligent system or for two or more intelligent systems to share knowledge in support of cooperative problem solving.
MACIP [Fan1999] [Cao1999b] [Cao1999]	National CIMS Research and Engineering Centre, Tsinghua Univ., P. R. China	CIMS Application Integration Platform (MACIP) is developed to offer manufacturing enterprises with a complete solution for the CIMS implementation through integrating a set of application software products. Operational Administration System (OAS) is the kernel of the MACIP to implement integration functions. Multi-agent technology is used in OAS to implement the integration of different software applications. Each agent is wrapped with one or more applications and takes these applications as services that can be provided to other agents. The communication and cooperation among these applications are implemented via service discovery among the agents. Applications may be added to or removed from the system at run time. Agents must be flexible enough to adapt to these dynamic behaviours of the system.

Table 3.1 Overview of Multi-Agent Systems: Applications and Tools

In the sections below, a coarse division of research topics that arise from the implementation of multi-agent systems is given. Each agent in the system is an autonomous entity with its own functions, data, resource, and environment. The basic characteristic of an agent is to manage its internal data at a knowledge level. In MACIP, an agent has knowledge about services provided by other agents and stores them in different tables. On the basis of knowledge representation, agents may also communicate with each other at a knowledge level. KQML can be used as an ACL for agents to exchange information and knowledge. Two agents may communicate on the same subject a number of times. Agent negotiation is discussed in detail in the ADEPT project and has been used successfully for business process management. Further relations among multiple agents can be characterised as agent coordination issues. JATLite provides one coordination model for multi-agent systems. These are also illustrated in Figure 3.1 and discussed further below.

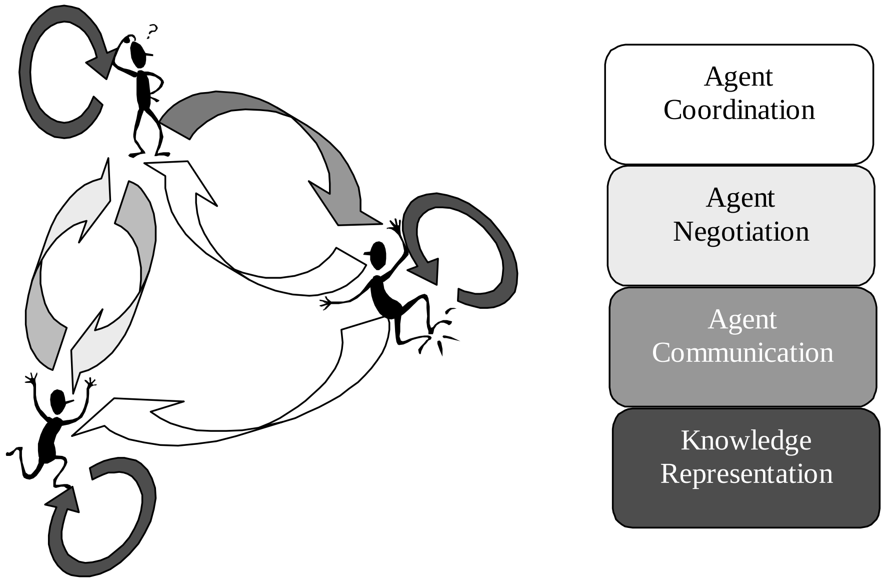


Figure 3.1 Research Topics in Multi-Agent Systems

3.1.1 Knowledge Representation

The knowledge representation of an agent is a correspondence between the external application domain and an internal symbolic reasoning system. The symbolic reasoning system is the agent's model of the external world and consists of data structures for storing information and procedures for manipulating these data structures. The mapping between the elements of the application domain and those of the domain model allows the agent to reason about the application domain by performing reasoning processes in the domain model, and transferring the conclusions back into the application domain.

As illustrated in Figure 3.2, in order to find a solution to a problem P in the application domain, this problem is first represented as P_m in the agent's domain model. Next the agent looks for a solution S_m of P_m in its domain model. Then the obtained solution S_m is reverse-mapped into S, which is the solution of the problem P.

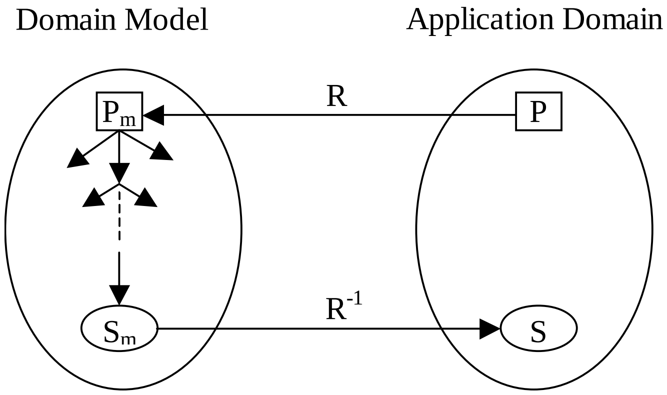


Figure 3.2 Knowledge Representation

The general features of a knowledge representation include representational adequacy, inferential adequacy, problem-solving efficiency, and learning efficiency. Good knowledge representation can lead to efficient knowledge reasoning, acquisition, and learning. More information on building knowledge-based agents can be found in [Tecuci1998].

3.1.2 Agent Communication

Agents usually interact by exchanging complex symbolic information and possibly have to agree on complex interaction protocols. In addition, agents are autonomous, possibly designed separately at different times by different people, and including heterogeneous software components. These issues led to the development of ACLs, such as KQML. A good summary on the many years of research into ACLs can be found in [Singh1998].

3.1.3 Agent Negotiation

Negotiation is the process by which two agents come to a mutually acceptable agreement on some matter. For an agent to influence an acquaintance, the acquaintance needs to be convinced that it should act in a particular way. The means of achieving this state are to make proposals, trade options, offer concessions, and (hopefully) come to a mutually acceptable agreement. More information on agent negotiation can be found in [Jennings2001, Kraus1998].

Though knowledge representation, agent communication and negotiation are important issues in the implementation of multi-agent systems, they are not the key consideration in the work described in this thesis. Our agent-based methodology designed for grid resource management system development focus more on agent coordination in a large scale.

3.1.4 Agent Coordination

Although ACLs and middleware systems, notably CORBA, are important to achieve interoperability, they mainly focus on peer-to-peer communications and do not account for a more comprehensive view of the interaction as a primary component of agents' societies. Therefore, both ACLs and middleware systems have to somehow be extended in order to include not only language and protocol specifications but also the definition of coordination laws, to allow for a global understanding and the management of interactions.

When a multi-agent system is made up of a large number of independently designed components, it may be very difficult to correctly design and manage the system as a whole. An approach that simply puts components together and lets them interact is likely to degenerate into chaos. Instead, models and tools are needed to put components together in a structured way. As already recognised in the area of software engineering, the design and management of a large software project requires the definition and analysis of its software architecture [Garlan1993, Perry1992]. This includes defining the role of each component, the mechanisms on which composition can be based, and their composition laws. A similar approach would be also helpful in the context of multi-agent systems. However, in this case, a more dynamic and flexible definition of the software architecture, that is interaction-oriented rather than composition-oriented, is needed.

Coordination is the art of managing interactions and dependencies among activities [Malone1994], that is, in the context of multi-agent systems, among agents. A coordination model provides a formal framework in which the interaction of software agents can be expressed [Gelernter1992]. A coordination model consists of three elements: the coordinables, the coordination media, and the coordination laws [Ciancarini1996]. Coordination models can be classified as control-driven or data-driven [Papadopoulos1998], which are also illustrated in Figure 3.3 and explained in detail below.

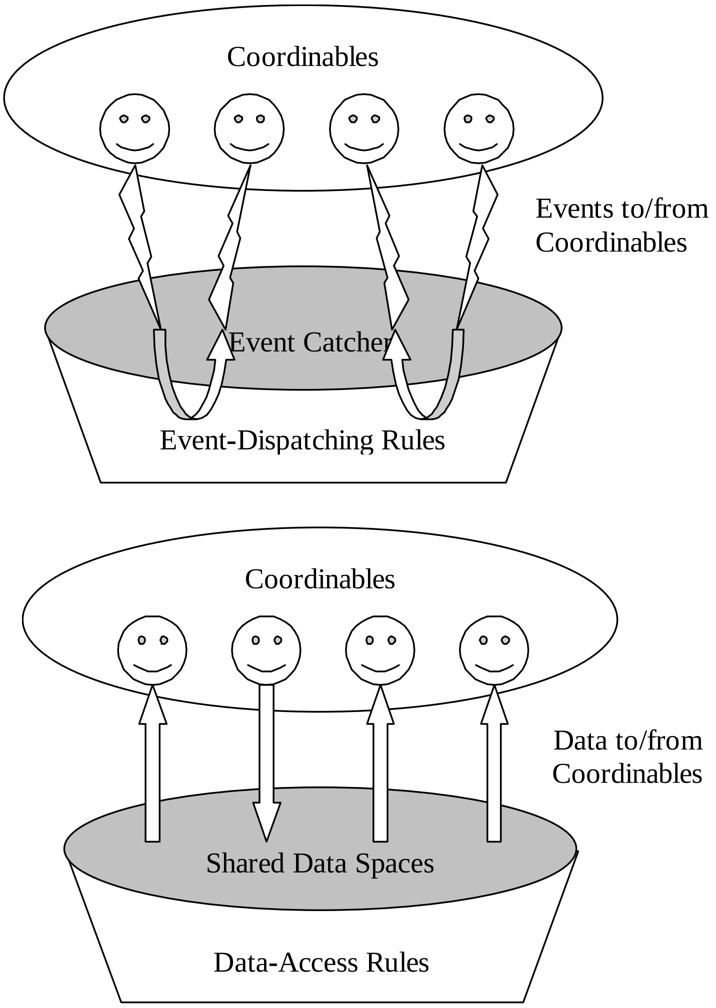


Figure 3.3 Coordination Models: Control-driven vs. Data-driven

- Control-driven. Coordinables (agents) typically open themselves to the external world and interact with it through events occurring on well-defined input/output ports. Manifold [Arbab1993] is a typical language that implements a control-driven coordination model.

- Data-driven. Coordinables interact with the external world by exchanging data structures through the coordination media, which especially acts as a shared data space. The research on data-driven coordination models originates from the parallel programming language Linda [Carriero1989].

Different application contexts exhibit different needs with respect to coordination, and the choice of a coordination model is likely to have a great impact in the design of multi-agent systems. In general, control-driven coordination models better suit those systems made up of a well-defined number of entities in which the flow of control and the dependencies between the components have to be regulated, and in which data exchange is not so important. The data-driven model on the other hand seems to better suit open applications, where a number of possibly pre-unknown and autonomous entities have to cooperate. In this case, the control driven model would somehow clash with the autonomy of the components and the dynamics of the open environment. Focusing on data preserves autonomy and dynamics of the components, which are usually designed to acquire information rather than control.

The grid environment is open and highly dynamic. The methodology developed to implement grid resource management adopts an extended data-driven mechanism for agents to exchange service information and cooperate with each other for service discovery.

In this section, we provide a brief introduction to multi-agent technologies. There is more than ten years of development of agent technologies. Agent-oriented software engineering [Wooldridge1999, Ciancarini2001] is emerging as another important approach complementing the structural method [Cao1996] and object-oriented method [Fan2000], especially in the case when more and more distributed software applications are emerging with increasing complexity and flexibility [Cao1999c]. A more detail introduction to theories, applications, methods, and tools of multi-agent systems can also be found in [Fan2001].

3.2 Service Advertisement and Discovery

As already stated, resource advertisement and discovery is an important issue in the implementation of grid resource management. In this section we will introduce service advertisement and discovery technologies for mobile computing. Many ideas described in this section can be applied directly to problems of resource discovery for grid computing. Table 3.2 gives an overview of six distributed system infrastructures with service discovery capabilities. A good survey can also be found in [Richard2000].

Name	Unit	Description	Service Discovery
Bluetooth [Bray2000] [Miller 1999]	IBM, Intel, Nokia, Ericsson, Toshiba	The Bluetooth protocols allow for the development of interactive services and applications over interoperable radio modules and data communication protocols.	The Bluetooth Service Discovery Protocol (SDP) provides a means for client applications to discover the existence of services provided by server applications as well as the attributes of those services. The attributes of a service include the type or class of service offered and the mechanism or protocol information needed to utilise the service. Features: Registry Advertisement ✓ Discovery ✓ Interoperability ✓ Security
HAVi [Lea2001]	Grundig, Hitachi, Panasonic, Philips, Sharp, Sony, Thomson, Toshiba	Home Audio-Video interoperability is a specification for home networks of consumer electronics devices. Typical HAVi devices are digital audio and video products such as cable modems, set-top boxes, digital and Internet-enabled TVs, and storage devices such as DVD drives for audio and video content. As technology advances and becomes more affordable, other kinds of HAVi devices may appear, such as videophones and Internet phones, which will plug into home networks and should be able to communicate without the user having to program them.	The approach the HAVi Architecture has adopted is to utilise Self Device Describing (SDD) data, required on all devices. SDD data contains information about the device, which can be accessed by other devices. The SDD data contains, as a minimum, enough information to allow instantiation of an embedded Device Control Module. This results in registration of device capabilities with the HAVi Registry, allowing applications to infer the basic set of command messages that can be sent to the device. Features: ✓ Registry Advertisement Discovery ✓ Interoperability Security

Jini [Amold1999] [Jini1999]	Sun Microsystems	A Jini system is a distributed system based on the idea of federating groups of users and the resources required by those users. The overall goal is to turn the network into a flexible, easily administered tool on which resources can be found by human and computational clients. The focus of the system is to make the network a more dynamic entity that better reflects the dynamic nature of the workgroup by enabling the ability to add and delete services flexibly.	The heart of the Jini system is a trio of protocols called discovery, join, and lookup. A pair of these protocols, discovery/join, occurs when a device is plugged in. Discovery occurs when a service is looking for a lookup service with which to register. Join occurs when a service has located a lookup service and wishes to join it. Lookup occurs when a client or user needs to locate and invoke a service described by its interface type (written in the Java programming language) and possibly, other attributes. Features: ✓ Registry ✓ Advertisement ✓ Discovery ✓ Interoperability ✓ Security
Salutation [Pascoe2001]	The Salutation Consortium	The Salutation architecture is created to solve the problems of service discovery and utilisation among a broad set of appliances and equipment and in an environment of widespread connectivity and mobility.	The architecture provides a standard method for applications, services and devices to describe and to advertise their capabilities to other applications, services and devices and to find out their capabilities. The architecture also enables applications, services and devices to search other applications, services or devices for a particular capability, and to request and establish interoperable sessions with them to utilize their capabilities. Features: ✓ Registry ✓ Advertisement ✓ Discovery ✓ Interoperability Security
SLP [Guttman1999]	The IETF	The Service Location Protocol provides a scalable framework for the discovery and selection of network services. Using this protocol, computers using the Internet need	SLP supports a framework by which client applications are modelled as User Agents and services are advertised by Service Agents. A third entity, called a Directory Agent provides scalability to the protocol.

		little or no static configuration of network services for network based applications. This is especially important as computers become more portable, and users less tolerant or able to fulfil the demands of network system administration.	Features: ✓ Registry ✓ Advertisement ✓ Discovery ✓ Interoperability ✓ Security
UPnP [UPnP 200] [Goland1999]	Microsoft	Universal Plug and Play (UPnP) is architecture for pervasive peer-to-peer network connectivity of PCs of all form factors, intelligent appliances, and wireless devices. UPnP is a distributed, open networking architecture that leverages TCP/IP and the Web to enable seamless proximity networking in addition to control and data transfer among networked devices in the home, office, and everywhere in between.	Simple Service Discovery Protocol (SSDP), as the name implies, defines how network services can be discovered on the network. SSDP defines methods both for a control point to locate resources of interest on the network, and for devices to announce their availability on the network. SSDP eliminates the overhead that would be necessary if only one of these mechanisms is used. Features: Registry ✓ Advertisement ✓ Discovery Interoperability Security

Table 3.2 Overview of Distributed System Infrastructures with Service Discovery Capabilities

Service advertisement and discovery technologies enable device cooperation and reduce configuration problems, which is a necessity in increasingly mobile computing environment. The main features of the service discovery suites include: service registry, service advertisement, service discovery, and interoperability. These are introduced in the sections below.

3.2.1 Service Registry

When a new component enters into a distributed system, there is usually a registration procedure for it to contact other existing components in the system. This process can be described by service registry.

In Jini, to register service availability or to discover services, a service or client must first locate one or more lookup servers by using a multicast request protocol. This request protocol terminates with the invocation of a unicast discovery protocol, in which clients and services are used to communicate with a specific lookup service.

Unlike Jini, SLP can operate without directory servers. The presence of one or more directory agents can substantially improve performance, however, this is done by reducing the number of multicast messages and the amount of network bandwidth used. In active discovery, service agents and user agents multicast SLP requests or use DHCP to discover directory agents. When a directory agent is present, service agents and user agents use unicast communication to register their services and find appropriate services respectively. In the absence of directory agents, user agents multicast requests for services and receive unicast responses directly from the service agents that control the matching services. This tends to increase bandwidth consumption, but provides a simpler model, appropriate for small networks.

In the A4 methodology introduced in this work, there is no distinction between clients, servers, and go-betweens as seen in Jini and SLP. Each agent in the system functions as a client, a server, or a directory, which provides a simpler model as well as resulting in a high performance implementation.

3.2.2 Service Advertisement

After joining the system, the components in the system operating as service providers must advertise their services to other components, which is referred to as service advertisement.

In UPnP, there is no service registry process. However, when devices are introduced into a network, they directly multicast “alive” messages to control points. When they want to cancel the availability of their services, they send “byebye” messages. In SSDP, each service has three associated IDs – service type, service name, and location – which are multicast when services are advertised.

Jini uses Java's Remote Method Invocation (RMI) facility for all interactions between either a client or a service and the lookup server after initial discovery of the lookup server. Jini associates a proxy, or remote control object, with each service instance. A service advertises its availability by registering its object in one or more lookup servers.

In the A4 methodology, service advertisement only happens between nearby agents so that the system is scalable, the details of which will be introduced in Chapter 5.

3.2.3 Service Discovery

The components acting as service requestors will search for available services in the system. This is the kernel process, which is defined as service discovery.

Bluetooth is a wireless radio system, so there is no service registry or advertisement in Bluetooth. The Bluetooth SDP provides a simple API for enumerating the devices in range, and browsing available services. It also supports "stop" rules that limit the duration of searches or the number of devices returned. Client applications use the API to search for available services either by service class that uniquely identify types of devices, or by matching attributes.

Salutation managers function as service brokers; they help clients find needed services and let services register their availability. A client can use the slmSearchCapability() call to determine if Salutation managers have registered specific functional units. Once a functional unit is discovered, slmQueryCapability() can be used to verify that a functional unit has certain capabilities.

In the A4 methodology, many agents can take part in a service discovery process. A service discovery process can traverse the system for many steps until the discovery succeeds or is forced to stop. This mechanism has a scalable implementation, which is different from all of the above methods.

3.2.4 Interoperability

When a client component in the system finds an available server component, whether these two components can cooperate with each other directly is described as the problem of interoperability.

In Jini, to use a service, a device must first secure an instance of the proxy object for it. From a client point of view, the location of the service provided by this remote control object is unimportant, because the object encapsulates the location of the service and the protocol necessary to operate it.

Salutation managers fill a role similar to lookup servers in Jini, but they can also manage the connections between clients and services. After the connection is established, a Salutation manager can operate in several “personalities”, with or without further operations in the data stream.

Unlike higher level service discovery technologies such as Jini, Bluetooth’s SDP does not provide a mechanism for using discovered services – specific actions required to use a service must be provided by a higher level protocol. However, it does define a standard attribute ProtocolDescriptionList, which enumerates appropriate protocols for communicating with a service.

In the initial implementation of A4 systems, the protocols for communication among agents are pre-defined using simple data structures. Interoperability is supported in a simple way, which may need further extensions for practical large-scale applications.

Another important issue, which is not a key consideration in the A4 methodology, is the feature of security. For example, Jini depends on Java’s security model, which provides tools like digital certificates, encryption, and control over mobile code activities. The security issues will not be discussed in detail here. The A4 methodology focuses on simulation based quantitative performance evaluation and optimisation of service discovery in large-scale multi-agent systems, which cannot be found in other work.

3.3 Use of Agent Technologies in Grid Development

The use of service discovery in multi-agent systems provides a suitable high-level abstraction for grid resource management, which will be described in detail in Chapter 5 as the so-called A4 methodology. In this section, we give a brief introduction to the state-of-the-art in the use of agent technologies in grid development.

Software agents have been used in several grid projects, such as AppLeS, DPSS, and NetSolve. In these projects, agents are high-level abstractions of software entities, which usually act as resource or data brokers or representatives of grid users in the grid software infrastructure. An agent-based grid computing project can be found in [Rana2001]. In this work, an “Agent Grid” is described that integrates services and resources for establishing multi-disciplinary PSEs (Problem Solving Environments). Specialised agents contain behavioural rules, and can modify these rules based on their interaction with other agents, and with the environment in which they operate. The A4 methodology can also be applied for integrating multiple services and resources. Rather than using a collection of many predefined specialised agents, a hierarchy of homogenous agents is used in the A4 methodology, where agents can be reconfigured with special roles at running time.

As mentioned, agents can achieve autonomy through intelligence and social ability. Both of these features can be used in grid development. For example, a resource scheduler is an important entity in a grid resource management system. Due to the large search space, AI technologies will most likely be used to solve large-scale resource scheduling issues. The powerful high-level abstraction of multi-agent systems can also be used to solve some architectural problems arising in grid development. In this work, we use agents for grid resource management.

As summarised in [Buyya2000b], there are three different models for grid resource management architecture: hierarchical model, abstract owner model, and computational market/economy model. In the methodology provided by A4, the agent system is organised in a hierarchical manner, which is used to address scalability. Meanwhile, each agent also acts as an abstract owner of the grid resources, and the service discovery process is performed in a market based way. Making full use of capabilities that agents provide, our architectural model for grid resource management can capture the essence of all three of the existing models.

During the past two years, the research into agents and the grid have begun to converge. A key sign of this trend can be seen clearly at CCGrid 2001. At this conference on cluster computing and the grid, two keynote speeches, one main conference section, and one workshop focused on research into agent technologies. It is clear that more agent applications on grid computing will emerge during the next few years.

However, agents cannot do everything, and there is also a long way to go to put grid computing into practice. In this work, we provide a framework (including methodology, functionality, and corresponding tools) for agent-based resource management for grid computing. There are many gaps that remain and require further work for a full grid resource management system. For example, an agent-based grid resource management system should be able to cooperate with other grid services (e.g. those provided by the Globus toolkit). These are not discussed in detail here.

3.4 Summary

Multi-agent and service discovery technologies have been introduced in detail in this chapter, which provides the background of the A4 methodology presented in Chapter 5. There is little research into the performance of large-scale multi-agent systems, because there are seldom such kinds of agent applications. This research is motivated by the development of a grid resource management system, which is large-scale with highly dynamic behaviours.

By the use of advanced agent technology the development of the software infrastructure in the grid is sure to accelerate. At the same time, new applications with new requirements will also stimulate the emergence of new technologies for software agents.

From a view of software engineering, agents provide high-level abstractions to the system. To implement an agent, different techniques must be applied according to requirements from different agent applications. In the work described in the thesis, performance prediction capabilities are one of the key features for the agent implementation, which can be provided by PACE. In the following four chapters, the main parts of the work are introduced, beginning with a case study of the performance evaluation using the PACE toolkit.

Chapter 4

SWEEP3D: PERFORMANCE EVALUATION USING THE PACE TOOLKIT

The grid resource management system is introduced, beginning with previous work at Warwick, that is the PACE toolkit. In this chapter, we validate the PACE performance prediction capabilities using a new parallel application [Cao1999d] called Sweep3D - a complex benchmark for evaluating wavefront application techniques on high performance parallel and distributed architectures [Koch1992]. This benchmark is also being analysed by other performance prediction approaches including POEMS. The sections below contain a brief overview of Sweep3D, the model description of the application, and validation results on two high performance systems.

4.1 Overview of Sweep3D

The benchmark code Sweep3D represents the heart of a real Accelerated Strategic Computing Initiative (ASCI) application [Nowak1997]. It solves a 1-group time-independent discrete ordinates (Sn) 3D cartesian (XYZ) geometry neutron transport problem. The XYZ geometry is represented by a 3D rectangular grid of cells indexed as IJK. The angular dependence is handled by discrete angles with a spherical harmonics treatment for the scattering source. The solution involves two main steps: the streaming operator is solved by sweeps for each angle, and the scattering operator is solved iteratively.

A sweep (Sn) proceeds as follows. For one of eight given angles, each grid cell has 4 equations with 7 unknowns (6 faces plus 1 central); boundary conditions complete the system of equations. The solution is by a direct ordered solve known as a sweep from one corner of the data cube to the opposite corner. Three known inflows allow the cell centre to be solved producing three outflows. Each cell's solution then provides inflows to 3 adjoining cells (1 in each of the I, J, & K directions). This represents a wavefront evaluation in all 3 grid directions. For XYZ geometries, each octant of angles has a different sweep direction through the mesh, but all angles in a given octant sweep the same way.

Sweep3D exploits parallelism through the wavefront process. The data cube undergoes a decomposition so that a set of processors, indexed in a 2D array, hold part of the data in the I and J dimensions, and all of the data in the K dimension. The sweep processing consists of pipelining the data flow from each cube vertex in turn to its opposite vertex. It is possible for different sweeps to be in operation at the same time but on different processors.

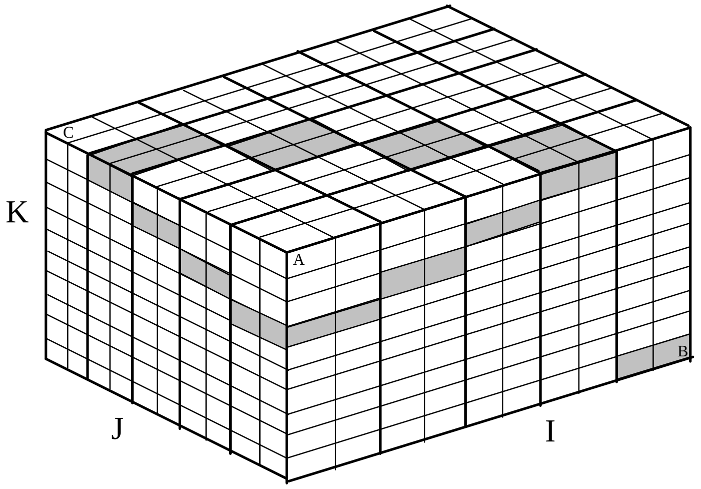


Figure 4.1 Data Decomposition of the Sweep3D Cube

For example, Figure 4.1 depicts a wavefront (shaded in Grey) that originated from the unseen vertex in the cube, and is about to finish at vertex A. At the same time, a further wavefront is starting at vertex B and will finish at vertex C. Note that the example shows the use of a 5x5 grid of processors, and in this case each processor holds a total of 2x2x10 data elements (data set of 10x10x10).

The version of Sweep3D that can be downloaded from the ASCI website is written entirely in Fortran77 except that it requires automatic arrays and a C timer routine is used. This version of Sweep3D supports both PVM [Geist1994] and MPI [Dongarra1994] message passing libraries as well as a single processor version. In this case study, we convert the Sweep3D programmes into a pure C version with only MPI functions, which can be used more conveniently for validation experiments of PACE performance modelling and prediction capabilities.

4.2 Sweep3D Models

In this section, we introduce the Sweep3D performance models in detail. The application model is composed of 9 objects written in the PACE PSL. The creation of resource models for two platforms is also introduced. The relations between the source code, application model, and resource model help a better understanding of the PACE methodology. The contents in this section correspond to those shown schematically in Section 2.2.

4.2.1 Model Description

We define the application object of the model as *sweep3d*, and divide each iteration of the application into four subtasks according to their different functions and different parallelisations. The object hierarchy is shown in Figure 4.2, each object is a separate rectangle and is labelled with the object name. The functions of each object are:

- *sweep3d* – the entry of the whole performance model. It initialises all parameters used in the model and calls the subtasks iteratively according to the convergence control parameter (*epsi*) as input by the user.

- *source* – subtask for getting the source moments, which is actually a sequential process.
- *sweep* – subtask for sweeper, which is the core component of the application.
- *fixed* – subtask to compute the total flux fixup number during each iteration.
- *flux_err* – subtask to compute the maximum relative flux error.
- *async* – a sequential “parallel” template.
- *pipeline* – parallel template specially made for the sweeper function.
- *globalsum* – parallel template which represents the parallel pattern for getting the sum value of a given parameter from all the processors.
- *globalmax* – parallel template which represents the parallel pattern for getting the maximum value of a given parameter from all the processors.
- *SgiOrigin2000* – contains all the hardware configurations for SGI Origin2000, which is comprised of smaller component hardware models already in existence within PACE. This can be interchanged with a hardware model of a different system, e.g. a cluster of Sun workstations.

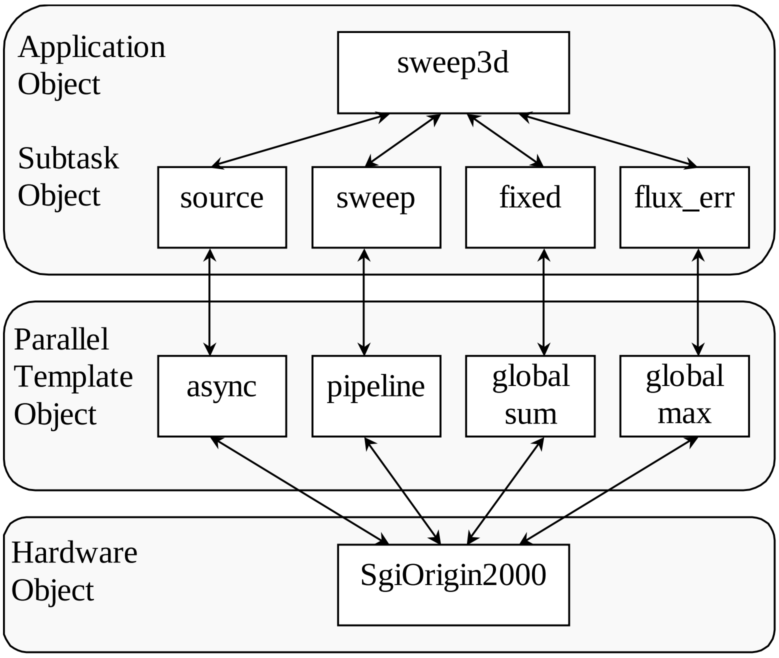


Figure 4.2 Sweep3D Object Hierarchy (HLFD Diagram)

4.2.2 Application Model Creation

The objects of application, subtask, and parallel template in the Sweep3D model introduced above can be expressed using the PACE PSL. The PSL code for Sweep3D is fully listed in Appendix A. Figure 4.3 describes different parts of the *sweep3d* object clearly in PSL scripts, the sections of which correspond to those schematically shown in Figure 2.2.

```
application sweep3d {
    include hardware;
    include source;
    include sweep;
    include fixed;
    include flux_err;
    var numeric:
        npe_i = 2,
        npe_j = 3,
        mk = 10,
        mmi = 3,
        it_g = 50,
        jt_g = 50,
        kt = 50,
        epsi = -12,
        ......
    link {
    hardware:
        Nproc = npe_i * npe_j;
    source:
        it = it,
        ......
    sweep:
        it = it,
    ......
    }
    option {
        hrduse = "SgiOrigin2000";
    }
    proc exec init {
        ......
        for(i = 1;i <= -epsi;i = i + 1) {
            call source;
            call sweep;
            call fixed;
            call flux_err;
        }
    }
}
```

Figure 4.3 Sweep3D Application Object

Each object follows the same syntax and requires the following parts:

- Include statement – declares other objects that are referenced.

- External variable definition – defines variables that form the interface to other objects as well as the PSL run-time system. The variables can be either numeric or strings.
- Linking statement – enables external variables and options defined in other objects to be modified.
- Option – sets the default options of the object.
- Procedures – describe the relationships between objects in order to predict performance. These relationships can either be described as control flow graphs (*cflow*) or execution statements (*exec*), which are analytical formulas. Each object also has a procedure *init*, which is the entry point for evaluation.

Some of the main statements used in the PSL to represent the performance aspects of the source code are as follows:

- *compute* – a processing part of the application, its argument is a resource usage vector. This vector is evaluated through the hardware object.
- *loop* – the body of which includes a list of the control flow statements that will be repeated.
- *call* - used to execute another procedure.
- *case* – the body of which includes a list of expressions and corresponding control flow statements which might be evaluated.
- *step* – corresponds to the use of one of the hardware resources of the system. Its argument is used to configure the device specified in the current step. This is used in parallel templates only.
- *confdev* – configures a device. The meaning of its arguments depend on the device. For example, the device *mpirecv* (MPI receive communication operation) accepts three arguments: source processor ID, destination processor ID and message size.

As mentioned before, application model creation can be processed almost automatically with the assistance of PACE application characterisation tool, which makes the performance modelling very easy and fast. However, during the performance modelling of Sweep3D, we still meet some difficulties.

Firstly, there are some aspects of the program that can be only processed by the PACE tools under guidance by the user. For example, the loop numbers in the program those are not explicit must be estimated by input from the user directly. PACE tools do not analyse data dependencies in the program. In Sweep3D, if a loop number is not a constant, we calculate an average value as an approximate estimation and input it to the model. The execution probabilities of each branch of *if* statements must also be estimated by the user, which make the implementation of PACE source code analysis tools much more efficient.

Secondly, there are some non-structural C statements like *goto* statement in the Sweep3D source code, which are not supported by PACE tools. We must give a reasonable estimation about these parts. Fortunately, those parts contain only a small number of instructions and have little impact on the overall execution time of the program.

Thirdly, *pipeline* is a parallel template specially made for the sweeper function, which is the kernel part of the Sweep3D model. Though, as mentioned before, a line by line mapping relation exists between the source code and corresponding parallel template, we must define the arguments of device configurations by ourselves, which need a deeper understanding of the parallelisation of Sweep3D. For example, the processors used by Sweep3D are logically organised into a 2D array, so the arguments for *mpirecv*, such as the source processor ID and the destination processor ID, must be calculated in advance. That is why *pipeline* looks much more complex than the other parallel template objects.

Though there are several approximate processes in the Sweep3D model, we can still get fairly reasonable performance prediction results given in the following sections. The accuracy of the performance prediction lies on not only the application model but also the hardware configurations described in the resource models.

4.2.3 Resource Model Creation

The resource models are embedded in the PACE tools ready for application performance evaluation. For ordinary usage, the PACE resource tools are not provided to the user. There are only a limited number of hardware platforms, so these models can be pre-installed into the PACE system, and be used directly for performance evaluation, which is convenient especially to those users who are not professional performance engineers. Figure 4.4 gives an illustration of part of the resource model for the multi-processor machine, the SGI Origin 2000.

```
config SgiOrigin2000 {

  hardware {
    ......
  }
  pvm {
    ......
  }
  mpi {
    ......
    DD_COMM_A = 512,
    DD_COMM_B = 33.228,
    DD_COMM_C = 0.02260,
    DD_COMM_D = -5.9776,
    DD_COMM_E = 0.10690,
    DD_TRECV_A = 512,
    DD_TRECV_B = 22.065,
    DD_TRECV_C = 0.06438,
    DD_TRECV_D = -1.7891,
    DD_TRECV_E = 0.09145,
    DD_TSEND_A = 512,
    DD_TSEND_B = 14.2672,
    DD_TSEND_C = 0.05225,
    DD_TSEND_D = -12.327,
    DD_TSEND_E = 0.07646,
    ......
  }
  clc {
    ......
  }
}
```

Figure 4.4 SGI Origin2000 Hardware Object

However, when a new hardware platform emerges, a new resource model should be produced for performance evaluation of applications running on this new resource. Also, if a new network API like MPI and PVM is developed, the corresponding configuration should also be added into each resource model. When we begin to evaluate the Sweep3D model on the SGI Origin2000, the MPI configurations shown in Figure 4.4 are actually not ready in the SGI Origin2000 model. In this section, we give a brief introduction to how this data is produced using PACE tools, which can lead to a deeper understanding of the working mechanisms of PACE.

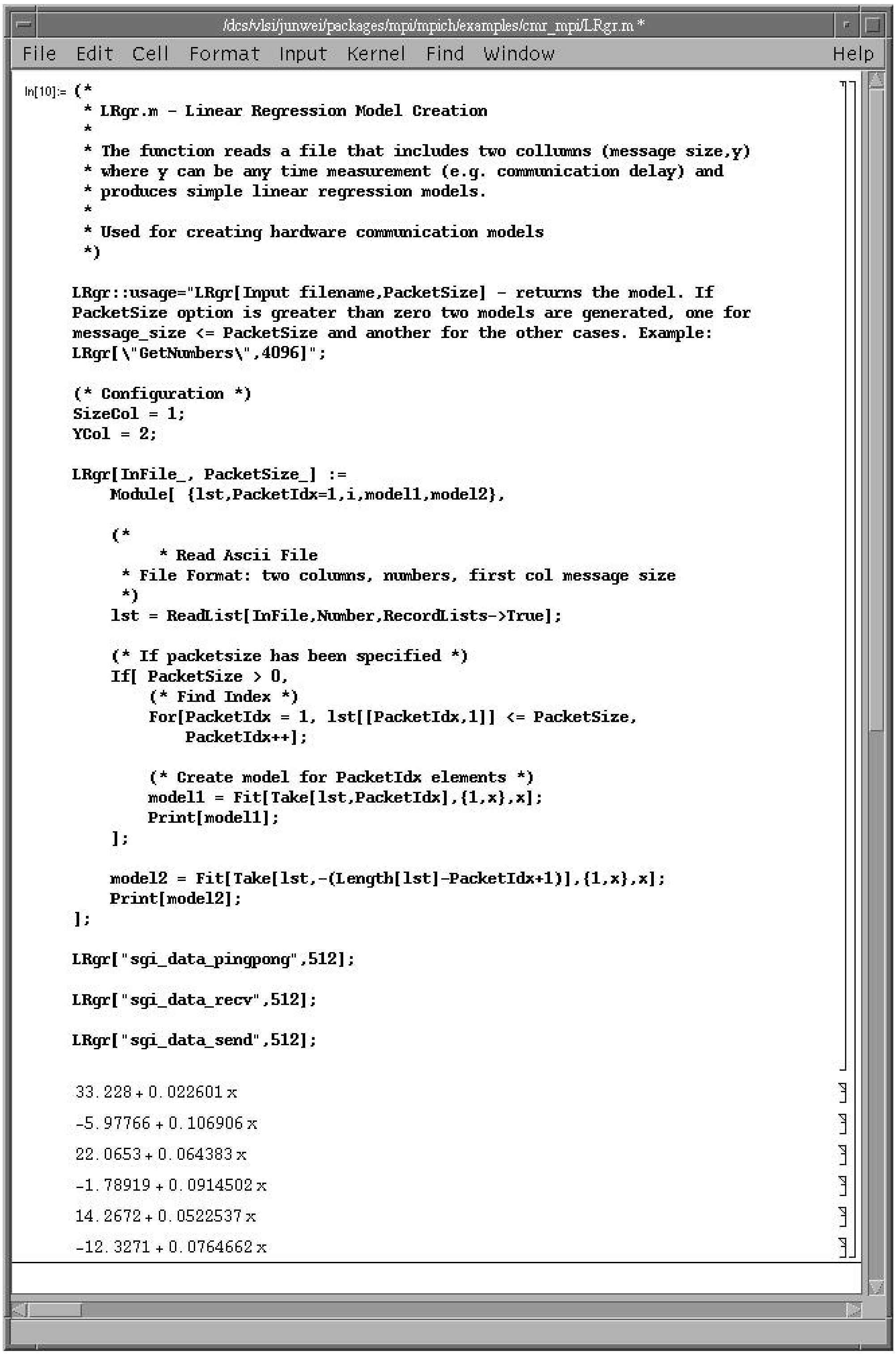
We notice that each MPI function is configured using five parameters, A to E. These parameters provide a simple description of MPI communications between processors of SGI Origin2000. They are used to calculate the consuming time of corresponding communication operation according to the follow equation:

$$T_x = \begin{cases} B + Cx\,, & if \qquad x \le A \\ D + Ex\,, & if \qquad x > A \end{cases}$$

where x is the number of double floats during one communication process (to make the evaluation of the Sweep3D model easy, we use the number of double floats directly as the variable. For general use of the model, x should be the number of communicating bytes).

A benchmark program with an MPI communication interface is run on two processors in a Ping-Pong style. For a given length of contents, the processors send it back and forth many times. Timers are added into the beginning and end points of the communication and measure the communication time consumed. Average values are calculated and recorded into the data files. In each data file, there are a number of data items. Each data item is a pair of data length and communication time.

Figure 4.5 gives a simple linear regression program written in Mathematica [Wolfram1991]. Given a data file, the function described in the program can calculate the five parameters and create corresponding hardware communication models. The results it produces from three data files are those parameters shown in Figure 4.4.


/dcs/vlsi/junwei/packages/mpi/mpich/examples/cmr_mpi/LRgr.m *
File Edit Cell Format Input Kernel Find Window Help
In[10]:=

```
(*
 * LRgr.m - Linear Regression Model Creation
 *
 * The function reads a file that includes two collumns (message size,y)
 * where y can be any time measurement (e.g. communication delay) and
 * produces simple linear regression models.
 *
 * Used for creating hardware communication models
 *)

LRgr::usage="LRgr[Input filename,PacketSize] - returns the model. If
PacketSize option is greater than zero two models are generated, one for
message_size <= PacketSize and another for the other cases. Example:
LRgr[\"GetNumbers\",4096]";

(* Configuration *)
SizeCol = 1;
YCol = 2;

LRgr[InFile_, PacketSize_] :=
    Module[ {lst,PacketIdx=1,i,model1,model2},

    (*
         * Read Ascii File
     * File Format: two columns, numbers, first col message size
     *)
    lst = ReadList[InFile,Number,RecordLists->True];

    (* If packetsize has been specified *)
    If[ PacketSize > 0,
        (* Find Index *)
        For[PacketIdx = 1, lst[[PacketIdx,1]] <= PacketSize,
            PacketIdx++];

        (* Create model for PacketIdx elements *)
        model1 = Fit[Take[lst,PacketIdx],{1,x},x];
        Print[model1];
    ];

    model2 = Fit[Take[lst,-(Length[lst]-PacketIdx+1)],{1,x},x];
    Print[model2];
];

LRgr["sgi_data_pingpong",512];

LRgr["sgi_data_recv",512];

LRgr["sgi_data_send",512];
```

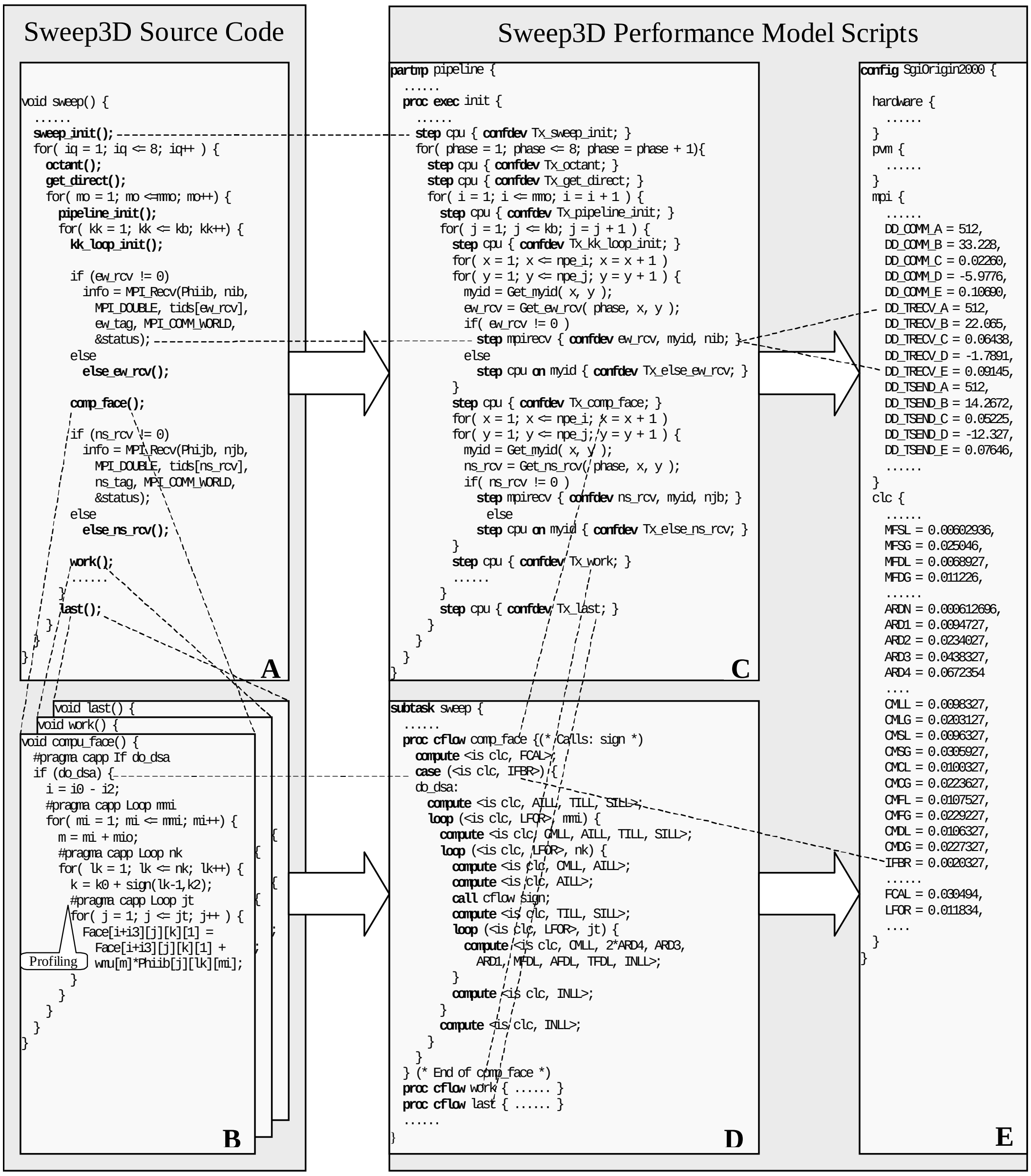
```
33.228 + 0.022601 x

-5.97766 + 0.106906 x

22.0653 + 0.064383 x

-1.78919 + 0.0914502 x

14.2672 + 0.0522537 x

-12.3271 + 0.0764662 x
```

Figure 4.5 Creating Hardware Communication Models Using Mathematica

PACE processor resource model creation will not be described here and can be found in [Papaefstathiou1994]. It is clear that the data included in the PACE resource models are static, which ignores the impact of the dynamic factors on the system performance, such as the changing of computing workload and communication bandwidth. For most of the tightly coupled parallel systems that are not overloaded, PACE resource models can still give good approximate and provide reasonable accuracy.

4.2.4 Mapping Relations

This section corresponds to those introduced in Section 2.2.4. The example model objects and their correspondence with the C source code are shown in Figure 4.6, which is a detailed example of Figure 2.5.



```
void sweep() {
  ......
  sweep_init();
  for( iq = 1; iq <= 8; iq++ ) {
    octant();
    get_direct();
    for( mo = 1; mo <=mmo; mo++) {
      pipeline_init();
      for( kk = 1; kk <= kb; kk++) {
        kk_loop_init();

        if (ew_rcv != 0)
          info = MPI_Recv(Phiib, nib,
            MPI_DOUBLE, tids[ew_rcv],
            ew_tag, MPI_COMM_WORLD,
            &status);
        else
          else_ew_rcv();

        comp_face();

        if (ns_rcv != 0)
          info = MPI_Recv(Phijb, njb,
            MPI_DOUBLE, tids[ns_rcv],
            ns_tag, MPI_COMM_WORLD,
            &status);
        else
          else_ns_rcv();

        work();
        ......
      }
      last();
    }
  }
}
```

```
void last() {
```

```
void work() {
```

```
void compu_face() {
  #pragma capp If do_dsa
  if (do_dsa) {
    i = i0 - i2;
    #pragma capp Loop mmi
    for( mi = 1; mi <= mmi; mi++) {
      m = mi + mio;
      #pragma capp Loop nk
      for( lk = 1; lk <= nk; lk++) {
        k = k0 + sign(lk-1,k2);
        #pragma capp Loop jt
        for( j = 1; j <= jt; j++ ) {
          Face[i+i3][j][k][1] =
            Face[i+i3][j][k][1] +
            wmu[m]*Phiib[j][lk][mi];
        }
      }
    }
  }
}
```

```
partmp pipeline {
  ......
  proc exec init {
    ......
    step cpu { confdev Tx_sweep_init; }
    for( phase = 1; phase <= 8; phase = phase + 1){
      step cpu { confdev Tx_octant; }
      step cpu { confdev Tx_get_direct; }
      for( i = 1; i <= mmo; i = i + 1 ) {
        step cpu { confdev Tx_pipeline_init; }
        for( j = 1; j <= kb; j = j + 1 ) {
          step cpu { confdev Tx_kk_loop_init; }
          for( x = 1; x <= npe_i; x = x + 1 )
          for( y = 1; y <= npe_j; y = y + 1 ) {
            myid = Get_myid( x, y );
            ew_rcv = Get_ew_rcv( phase, x, y );
            if( ew_rcv != 0 )
              step mpirecv { confdev ew_rcv, myid, nib; }
            else
              step cpu on myid { confdev Tx_else_ew_rcv; }
          }
          step cpu { confdev Tx_comp_face; }
          for( x = 1; x <= npe_i; x = x + 1 )
          for( y = 1; y <= npe_j; y = y + 1 ) {
            myid = Get_myid( x, y );
            ns_rcv = Get_ns_rcv( phase, x, y );
            if( ns_rcv != 0 )
              step mpirecv { confdev ns_rcv, myid, njb; }
                else
              step cpu on myid { confdev Tx_else_ns_rcv; }
          }
          step cpu { confdev Tx_work; }
          ......
        }
        step cpu { confdev Tx_last; }
      }
    }
  }
}
```

```
subtask sweep {
  ......
  proc cflow comp_face {(* Calls: sign *)
    compute <is clc, FCAL>;
    case (<is clc, IFBR>) {
    do_dsa:
      compute <is clc, AILL, TILL, SILL>;
      loop (<is clc, LFOR>, mmi) {
        compute <is clc, CMLL, AILL, TILL, SILL>;
        loop (<is clc, LFOR>, nk) {
          compute <is clc, CMLL, AILL>;
          compute <is clc, AILL>;
          call cflow sign;
          compute <is clc, TILL, SILL>;
          loop (<is clc, LFOR>, jt) {
            compute <is clc, CMLL, 2*ARD4, ARD3,
              ARD1, MFDL, AFDL, TFDL, INLL>;
          }
          compute <is clc, INLL>;
        }
        compute <is clc, INLL>;
      }
    }
  } (* End of comp_face *)
  proc cflow work { ...... }
  proc cflow last { ...... }
  ......
}
```

```
config SgiOrigin2000 {

  hardware {
    ......
  }
  pvm {
    ......
  }
  mpi {
    ......
    DD_COMM_A = 512,
    DD_COMM_B = 33.228,
    DD_COMM_C = 0.02260,
    DD_COMM_D = -5.9776,
    DD_COMM_E = 0.10690,
    DD_TRECV_A = 512,
    DD_TRECV_B = 22.065,
    DD_TRECV_C = 0.06438,
    DD_TRECV_D = -1.7891,
    DD_TRECV_E = 0.09145,
    DD_TSEND_A = 512,
    DD_TSEND_B = 14.2672,
    DD_TSEND_C = 0.05225,
    DD_TSEND_D = -12.327,
    DD_TSEND_E = 0.07646,
    ......
  }
  clc {
    ......
    MFSL = 0.00602936,
    MFSG = 0.025046,
    MFDL = 0.0068927,
    MFDG = 0.011226,
    ......
    ARDN = 0.000612696,
    ARD1 = 0.0094727,
    ARD2 = 0.0234027,
    ARD3 = 0.0438327,
    ARD4 = 0.0672354
    ....
    CMLL = 0.0098327,
    CMLG = 0.0203127,
    CMSL = 0.0096327,
    CMSG = 0.0305927,
    CMCL = 0.0100327,
    CMCG = 0.0223627,
    CMFL = 0.0107527,
    CMFG = 0.0229227,
    CMDL = 0.0106327,
    CMDG = 0.0227327,
    IFBR = 0.0020327,
    ......
    FCAL = 0.030494,
    LFOR = 0.011834,
    ....
  }
}
```

Figure 4.6 Mapping between Sweep3D Model Objects and C Source Code

Figure 4.6A is the C source code showing part of the main function *sweep*, whose serial parts have been abstracted into a number of sub-functions in bold font. Figure 4.6C shows how the same source code structure is used to provide the parallel template description. Figure 4.6B is an example sub-function source code, which can be converted automatically to the control flow procedure in the subtask object as shown in Figure 4.6D.

Figure 4.6 also shows the inner mapping between the software objects and hardware object of the performance model. All of the performance specification components in PSL can find their corresponding configurations from the hardware object, shown in Figure 4.6E. The abundant off-line configuration information included by the hardware object is the basis to implement a rapid evaluation time to produce the performance predictions.

It can be seen from the part of the Sweep3D model that there is a lot of information extracted from the source code that is used for the performance prediction. The accuracy of the resulting model is of importance, and in Section 4.3 below, detailed results are shown to validate the model with measurements on the two systems considered.

4.3 Validation Experiments

In this section validation results on execution time for Sweep3D are given to illustrate the accuracy of the PACE modelling capabilities for performance evaluation. The procedures in the PACE evaluation engine to achieve these results have been introduced in Section 2.5.

4.3.1 Validation Results on SGI Origin2000

Table 4.1 shows the validation results of the PACE model against the code running on an SGI Origin2000 shared memory system. Note that the result for single processor input are not included because there are many special configurations, which are not included in the current performance model for the sequential code. The accuracy of the performance prediction results were evaluated as follows:

$$\text{Error} = \frac{\text{Prediction - Measurement}}{\text{Measurement}} \times 100\%$$

The errors between measurements and predictions are also shown in Table 4.1. It can be seen that the maximum error is 11.44%, but the average error is approximately 5%.

Data Size	2D Proc. Array	Total Time		
		Prediction (s)	Measurement (s)	Err (%)
15x15x15	1x2	4.73037	4.440255	6.53
	2x2	2.59659	2.584936	0.45
	2x3	1.8373	1.812252	1.38
	2x4	1.51869	1.609818	-5.66
	3x3	1.3399	1.343736	-0.29
	3x4	1.10918	1.164072	-4.72
	4x4	0.907100	1.002728	-9.54
25x25x25	1x2	22.9501	20.780170	10.44
	2x2	12.1537	11.619632	4.60
	2x3	7.83574	7.893481	-0.73
	2x4	6.02865	5.979522	0.82
	3x3	5.52498	5.532116	-0.13
	3x4	4.24959	4.469564	-4.92
	4x4	3.36453	3.537966	-4.90
35x35x35	1x2	69.3858	64.832165	7.02
	2x2	36.1978	33.097098	9.37
	2x3	22.1074	21.160975	4.47
	2x4	16.3181	16.137180	1.12
	3x3	15.3466	15.272606	0.48
	3x4	11.3211	11.451001	-1.13
	4x4	8.84226	9.984213	-11.44
50x50x50	1x2	217.398	228.893311	-5.02
	2x2	112.307	102.285787	9.80
	2x3	65.6201	67.278086	-2.46
	2x4	46.7591	49.534483	-5.60
	3x3	45.1373	47.289627	-4.55
	3x4	32.1438	34.796392	-7.62
	4x4	24.8468	24.800020	0.20

Table 4.1 PACE Model Validation on an SGI Origin2000

The validation results are also illustrated in Figure 4.7. As shown in the figure, run time decreases when the number of processors increases. At the same time the parallel efficiency decreases too. In fact when the number of processors is more than 16, the run time does not improve any further.

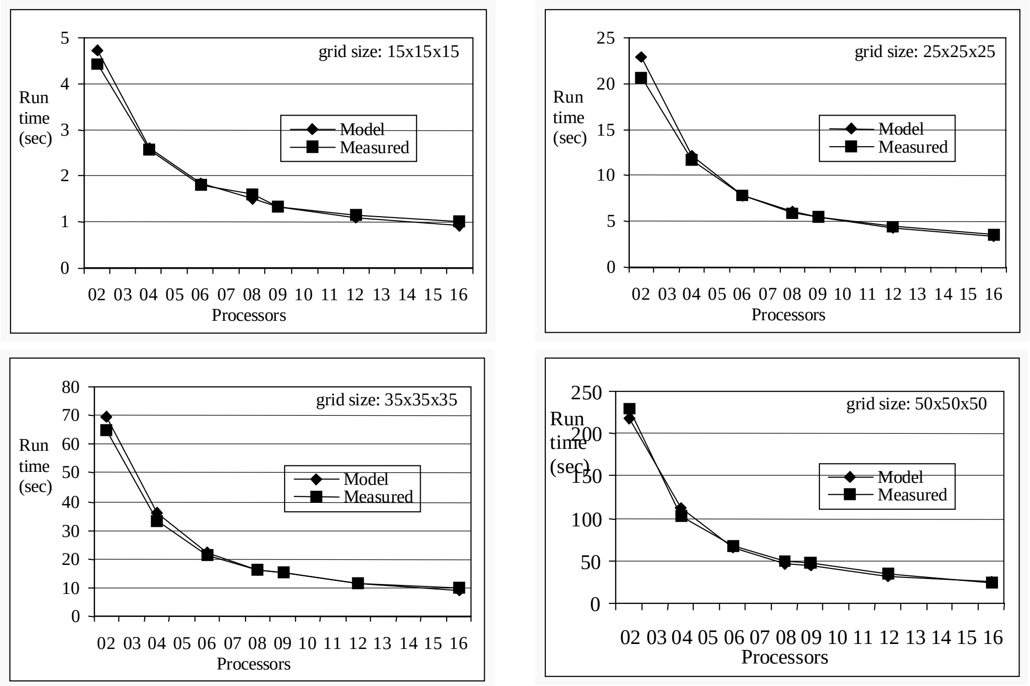


Figure 4.7 PACE Model Validation on an SGI Origin2000

4.3.2 Validation Results on Sun Clusters

By only changing the hardware object to the SunUltra1 predictions on this new system can be obtained as shown in Table 4.2. A cluster of 9 SunUltra1 workstations was used to obtain the measurements assuming no background loading.

Data Size	2D Proc. Array	Total Time		
		Prediction (s)	Measurement (s)	Err (%)
15x15x15	1x2	11.597	12.442062	-6.79
	2x2	7.42898	6.938457	7.07
	2x3	5.88532	5.659182	4.00
	2x4	5.29021	5.445188	2.85
	3x3	4.84622	5.101984	5.01
25x25x25	1x2	51.4059	51.326475	0.15

	2x2	29.6231	27.409842	8.07
	2x3	20.5203	20.188288	1.64
	2x4	16.7535	17.007142	-1.49
	3x3	15.5563	15.041854	3.42
35x35x35	1x2	149.708	145.008424	3.24
	2x2	82.8056	78.401377	5.62
	2x3	53.097	53.201457	-0.20
	2x4	40.9785	42.817732	-4.30
	3x3	38.4032	37.551111	2.27
50x50x50	1x2	456.928	462.103560	-1.12
	2x2	244.501	232.202359	5.30
	2x3	147.7	147.227193	0.32
	2x4	108.571	120.719472	-10.06
	3x3	103.838	104.700557	0.82

Table 4.2 PACE Model Validation on a Cluster of SunUltra1 Workstations

It can be seen that the maximum error is 10.06%, but the average error is also approximately 5%. As shown in Figure 4.8, the run time spent is much more than that on SGI Origin2000 with the same workload. But the trend of the curve is almost the same.

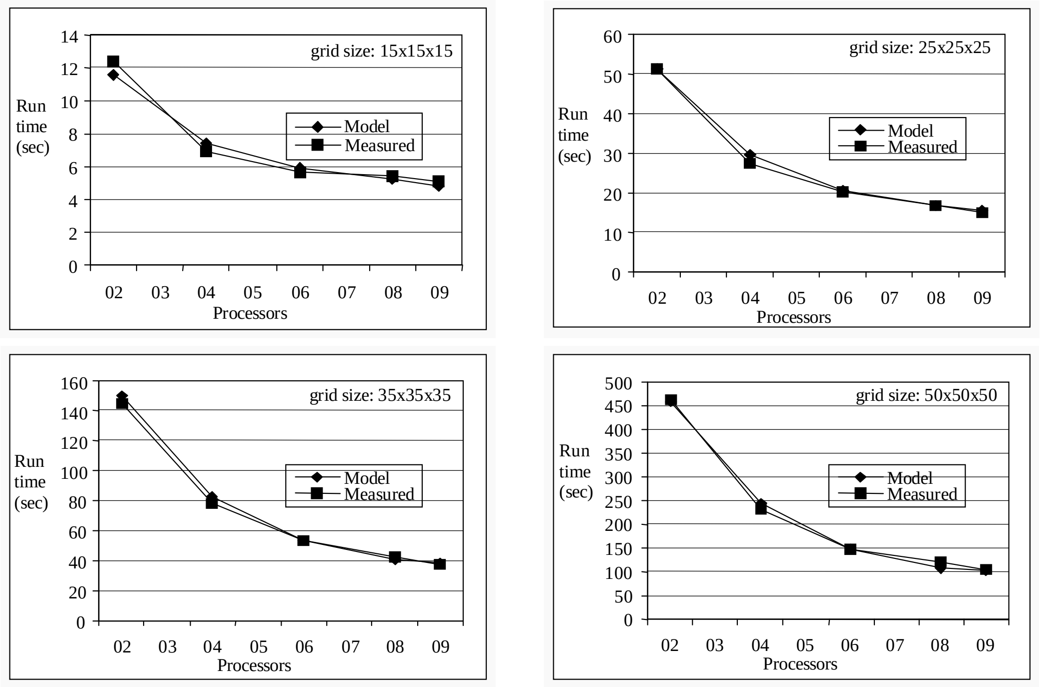


Figure 4.8 PACE Model Validation on a Cluster of SunUltra1 Workstations

Besides the reasonable accuracy, the performance model can be used to obtain the evaluation results in a rapid time period, typically less than 2s. This is a key feature of PACE that enables the performance models to be used to steer the application execution onto an available system at run-time in an efficient manner [Kerbyson1998, Alkindi2001].

4.4 PACE as a Local Resource Manager

In this chapter, we use Sweep3D as a case study to validate the performance prediction capabilities of the PACE toolkit. The key features of PACE performance prediction capabilities include:

- a reasonable prediction accuracy (the maximum error between measurements and predictions is 15%);
- a rapid evaluation time (typically seconds of CPU use) for a given system and problem size;
- and easy performance comparison across different computational systems.

It has been shown that the PACE system can produce reliable performance information which may be used for investigating application and system performance in many different ways. As mentioned in [Kerbyson2000], performance data produced by PACE can be used for the management of parallel and distributed systems. However, the PACE toolkit is initially not developed in the context of grid computing. In this section, we will discuss whether PACE functions can be used to produce performance related data for resource management in a grid environment.

As we have mentioned in Section 2.4, a grid environment brings two key challenges, which are scalability and adaptability. For the grid resource management system to be scalable, it is obviously not possible to provide the whole grid resources with one PACE manager. In this case, it will definitely become the bottleneck of the system. It is practical that one PACE resource manager may be able to manage and schedule applications running on a local resource.

PACE models contain only static information on the application and system. PACE application model is retrieved directly from the source code of the parallel application. The hardware information contained in PACE resource models is measured off-line on computing and communication capabilities of the resource. When a parallel application is executed on a grid resource, there are many dynamic factors that have an impact on the resource performance. For example, the grid resource may not be entirely dedicated to the grid users. Especially the communication between the grid resources is provided by low speed networks, which may result in irregular communication latency when parallel applications are running. PACE prediction will not provide the same reasonable accuracy under such kind of highly dynamic situation.

In summary, while extremely well suited for managing a locally distributed multi-computer, the PACE functions do not map well onto wide-area grid computing environments, where heterogeneity, multiple administrative domains, and communication irregularities dramatically complicate the job of resource management.

As illustrated in Figure 2.8, grid resource management functions should be performed at both local and meta levels. Our method for grid resource management is to use PACE as local resource manager. At the meta level, an additional mechanism, summarised as the A4 methodology in the following chapter, are introduced to coordinate different local resource managers to achieve the overall management of grid resources.

Chapter 5

A4: AGILE ARCHITECTURE AND AUTONOMOUS AGENTS

A4 (Agile Architecture and Autonomous Agents) is a methodology for building large-scale distributed software systems with highly dynamic behaviours [Cao2001c]. The methodology is developed in order to be used for meta-level grid resource management, which is an extension of work described in [Cao2000b]. A4's emphasis is on dealing with architectural level dynamics and using simulation based analysis to provide quantitative performance evaluation and optimisation of system behaviours, which differentiate A4 from other distributed system infrastructures described in Section 3.2.

- An agent is the main component in the system. Each has its own motivation, resource and environment. They are not predetermined to work together. The number of agents will dramatically increase when a wide-area software environment is considered. Together they form a large-scale multi-agent system.
- Autonomy is used to describe the character of the agent. The autonomy is mainly achieved by the intelligence and the social ability of the agents. An agent can fulfil high-level tasks by its own intelligence or by cooperating with other agents continuously with little human interference.

- Architecture is used to provide a glue for the interactions between the agents. For example, large-scale multiple agents can be organized into a hierarchy.
- Agility is used to describe the character of the architecture. Agility means quick adaptation to environmental change. Autonomy provides the system with component-level adaptability, while agility provides the architecture-level adaptability of the system.

5.1 Agent Hierarchy

The hierarchical model is illustrated in Figure 5.1. There is a single type of component, the agent, which is used to compose the whole system. Each agent has the same set of functions. Agents are organised into a hierarchy. In Figure 5.1 different terms are used to differentiate the level of the agent in the hierarchy. The broker is an agent that heads the whole hierarchy, maintaining all service information of the system. A coordinator is an agent that heads a sub-hierarchy. A leaf-node is actually termed an agent in this description.

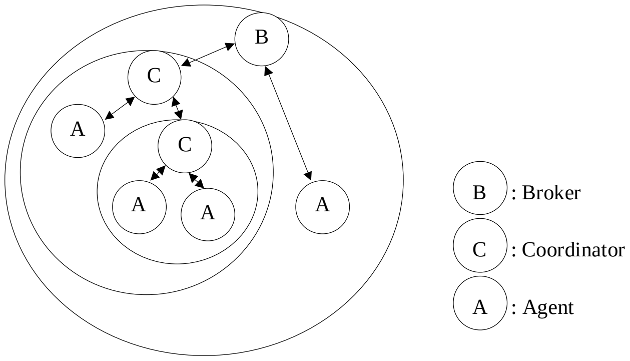


Figure 5.1 Agent Hierarchy

The broker and coordinators are also agents except that they are in a special position in the hierarchy. All the agents have the same function despite their different positions. The broker does not have any more priorities than coordinators or agents. The hierarchy of homogenous agents gives a high-level abstraction of a distributed system.

The agent hierarchy can also represent an open and dynamic system. New agents can join the hierarchy or existing agents can leave the hierarchy at any time. When a new agent wants to join the system, in the hierarchical model, it will broadcast to find its nearest existing agent. An agent can only have one connection to an agent higher in the hierarchy to register with, but be registered with many lower level agents. Each agent records related registration information. After registration, agents can communicate with each other using unicast instead of multicast. When an agent wants to leave the system, it must contact its upper agent to cancel the registration, and also inform its lower agents to re-register in the hierarchy.

The hierarchy model can address partly the problem of scalability. When the number of agents increases, the hierarchy may lead to many system activities being processed in a local domain. In this way the system may scale well and does not need to rely on one or a few central agents, which may otherwise become a system bottleneck.

Service is another important concept in the A4 methodology. Request is a complementary concept to service. In other methodologies, a client is abstracted into a request sender; a server is abstracted into a service provider; and a matchmaker is an abstraction of a router between a client and corresponding server. In the A4 methodology, an agent contains all of the above abstractions. An agent can send requests and provide services. Every agent can act as a router between a request and a service. This gives a simple and uniform abstraction of the functions in the system.

A resource can be a program, a device or a human in the system, where a service is originally provided, while a user is a human, where a request is originally sent out. An agent can be a manager of one or more resources. When a resource is available to provide a service, the corresponding agent is responsible for distributing the service information to many other agents. When a user wants to send a request, it usually finds and contacts its nearest agent, and a request may pass by many agents to reach the required resource. These processes that happen in the agent hierarchy are defined as service advertisement and discovery, which will be discussed in detail in the following sections.

5.2 Agent Structure

The agent hierarchy gives an overall architectural description of the system. In this section, a layered agent structure is considered, which can provide functions both for local management and global coordination. The structure is illustrated in Figure 5.2 and explained in detail below.

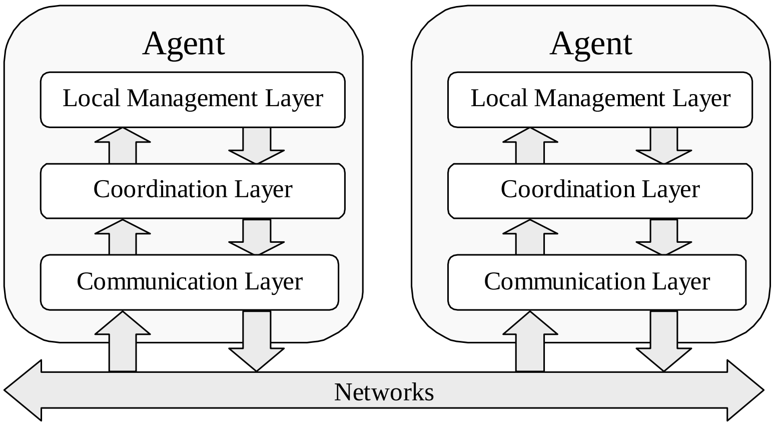


Figure 5.2 Layered Agent Structure

- Communication Layer – Agents in the system must be able to communicate with each other using common data models and communication protocols. ACL can be used to address these problems. However, an initial system implementation can use some simple pre-defined data structures instead of a language. The communication layer provides an agent with an interface to heterogenous networks and operating systems.
- Coordination Layer – The request an agent receives from the communication layer should be explained and submitted to the coordination layer, which decides how the agent should act on the request according to its own knowledge. For example, if an agent receives a service discovery request, it must decide whether it has related service

information. Our methodology focuses on the implementation of this layer.

- Local Management Layer – This layer encapsulates the functions needed for local system management. For example, if an agent finds that the required service is within its own capabilities, the request will be submitted to this layer from the coordination layer to access the resource. This local manager can also provide service information to the coordination layer. Different agents can include different functions for local system management.

How the agents in the system cooperate with each other is up to the functions implemented in the coordination layer in each agent. In the A4 methodology, these functions are described as two complementary processes, service advertisement and discovery, which will be described in detail below.

5.3 Service Advertisement

An agent in the system can have many local resources that can provide services. The agent can take them as its own capabilities. Local management in an agent is responsible for collecting this service information and provide it to the coordination layer, where this information is stored. An agent must decide how and when to advertise this service information to other nearby agents.

An agent can also receive many service advertisements from nearby agents and also store this information in its coordination layer as its own knowledge. All of the service information are organised into Agent Capability Tables (ACTs).

5.3.1 Agent Capability Tables

An ACT item is composed of three constituent parts:

- Agent ID. This ID includes the contact information of an agent. During the registration process described before, an agent can only get ID information and contact its upper or lower agents. With the agent IDs stored in ACTs, an agent can also contact more agents and cooperate with them for service discovery.

- Service Information. Service information should contain all performance related information about a resource. This information will be used by the agent to evaluate the performance of corresponding resources, estimate the capability of corresponding agents, and make service discovery decisions. In general, a name should be defined for each service.
- Options. Additional options can be added into each ACT item to constrain agent behaviours for service advertisement and discovery. Concrete options will be introduced in detail later.

When a new resource is available to provide service, its agent should advertise the service information to other agents. The performance of services offered by an agent can change over time. When this occurs, the corresponding service information needs also to be updated. When a service becomes unavailable, it needs to advertise to cancel previous information that has been advertised into the hierarchy. The dynamics of the system increase the difficulty of system management.

An agent can choose to maintain different kinds of ACTs according to different sources of service information. These include:

- T_ACT (This ACT). In the coordination layer of each agent, T_ACT is used to record service information of local resources. The local management layer is responsible for collecting this information and reporting it to the coordination layer.
- L_ACT (Local ACT). Each agent can have one L_ACT to record the service information received from its lower agents. The services recorded in L_ACT are provided by the resources in its local scope.
- G_ACT (Global ACT). The G_ACT in an agent is actually a record of the service information received from its upper agent. The service information

recorded in G_ACT are provided by the agents, which have the same upper agent as the agent itself.

- C_ACT (Cached ACT). Cached service information is stored in C_ACT. When an agent sends a request for service discovery, the returned result can be stored in C_ACT, and hence looked up when next requested.

5.3.2 ACT Maintenance

The performance of the resources that provide services may vary over time, which may cause the corresponding service information that is stored in the ACTs of other agents to become out-of-date. There are basically two ways to maintain the contents of ACTs in an agent: data-pull and data-push, each of which have two approaches: periodic and event-driven. These are summarised in Table 5.1.

Type	Approach	ACT	Description
Data-pull	Periodic	T_ACT	The ACT management can ask local management to monitor its own resources and return the service information to T_ACT periodically.
		L_ACT	An agent can ask its lower agents for the service information they have, and update its own L_ACT periodically.
		G_ACT	An agent can ask its upper agent for the service information it has, and update its own G_ACT periodically.
		C_ACT	An agent can check whether the service information in its cache is out-of-date periodically. Any unavailable service information will be deleted.
	Event-driven	T_ACT	A service discovery process can trigger a T_ACT updating. When a request arrives and an agent looks up the T_ACT, the ACT management can ask local management to monitor its own resources and return the service information to T_ACT immediately.
		L_ACT	When a request arrives, an agent can ask its lower agents for the service information they have, and update its own L_ACT immediately.
		G_ACT	When a request arrives, an agent can ask its upper agent for the service information it has, and update its own G_ACT immediately.
		C_ACT	When a request arrives, an agent can check whether the service information in its cache is still available. Any out-of-date service information will be deleted.

Data-push	Periodic	T_ACT	The local management in an agent can monitor its resources and submit the results to the ACT management in the coordination layer periodically.
		L_ACT	Lower agents can report their service information periodically to update the L_ACT of an agent.
		G_ACT	The upper agent can multicast its service information to its lower agents periodically to update their G_ACTs.
		C_ACT	N/A
	Event-driven	T_ACT	When a resource changes, the local management in the agent will inform the change to the T_ACT in the coordination layer of the agent immediately.
		L_ACT	When one service information changes in a lower agent, it will report the change to update the L_ACT of an agent immediately.
		G_ACT	When one service information changes in the upper agent, it will multicast the change to its lower agents immediately to update their G_ACTs.
		C_ACT	When a service discovery result is retuned to an agent, the agent can update its C_ACT immediately.

Table 5.1 Service Advertisement and ACT Maintenance

From the methods described above, it is clear that most of the service advertisement which occurs in an agent hierarchy happens only between nearby agents. An agent can only advertise its service information to its upper agent or lower agents. However, service information can also be spread to a large area after many steps of advertisement over a period of time. This is an important feature to make the system scalable and to avoid any communication bottlenecks. The same principles are also applied to the service discovery processes.

5.4 Service Discovery

Each agent has different kinds of ACTs maintained by service advertisement. An agent takes the contents in ACTs as its own knowledge, which is mainly used for service discovery. A service discovery process is triggered by the arrival of a request in an agent. A request is usually composed of several parts:

- Request information. These include details of services the user wants to discover. This information may combine with service information in ACTs to produce high-level performance information of corresponding resources.

- Requirement. This includes required performance information from the user, which may be used for matchmaking for agents to make decisions on whether a resource can provide a capable service or not.
- Options. Additional options may be attached with each request, which may include user control information for the service discovery. For example, the user may limit time and scope of a discovery process.

An agent can act on a request in a number of ways, for instance:

- Yes. I can provide required service, so the discovery ends successfully.
- No. I cannot provide the required service. However, I know an agent, which may have the capability to provide the required service. I can transfer the request to it for further discovery.
- No. I have no idea of the required service. However, I can transfer the request to lower or upper agents for further discovery.
- No. I have no idea of the required service, and there are also no other agents that I can query. I am sorry that the discovery has failed.

5.4.1 ACT Lookup

The process of service discovery in an agent is the process of looking up the ACTs. The general order for an agent to check different kinds of ACTs in turn is: T_ACT, C_ACT, L_ACT, and finally G_ACT, which will be explained one by one below.

An agent is a representative of its own resources in the large-scale environment. When an agent receives a request from a user or another agent, it is natural that it will check its own capabilities recorded in the T_ACT firstly. If an agent is aware that it can provide the required service itself, the service discovery is successful and the service information will be returned to where it came from.

If there is no required service information in the T_ACT, an agent may choose to look up its C_ACT. Previous service discovery results are cached in the C_ACT, which have more possibility to meet the requirements from the following requests. If required service information is found in C_ACT, the agent will check whether the service is still available. If so, the request will be dispatched to the corresponding agent. Otherwise, the agent will update the C_ACT and process other service discovery.

If there is no required service information in the C_ACT either, an agent may then choose to look up its L_ACT. L_ACT records service information in local scope. Most users prefer to find an available resource located as near as possible. So it is reasonable to check L_ACT first instead of the G_ACT. If the required service information is found in the L_ACT, the request will be dispatched to the corresponding agent. Otherwise, additional service discovery will have to be processed.

An agent can finally looks up its G_ACT. The G_ACT records service information in a much wider scope and provides opportunities to find the required service. If the required service information is found in G_ACT, the request will be dispatched to the corresponding agent. Otherwise, the agent must make decisions for the following action.

An agent may not maintain all of the above ACTs. T_ACT is generally maintained in each agent. If an agent does not choose to cache previous discovery results, there will be no need to look up the C_ACT. An agent can also choose not to maintain L_ACT or G_ACT. If there is no L_ACT information and an agent cannot find any information in its ACTs, it may choose to pass the request to one of its lower agents.

If an agent looks up all of the ACTs and still does not get the required service information, it may consider submitting the request to its upper agent. The upper agent will follow the same procedure, but may maintain service information in a larger scope, thus may be more possible to find an available service.

If an agent looks up all of the ACTs and does not get the required service information, and there is no other agent it can contact for further discovery, the service discovery ends as failed. For example, consider a broker that has no upper agent. If a request reaches the broker of the agent hierarchy and the broker fails to find required service information in its ACTs, the discovery has to end unsuccessfully.

From the above description, a service discovery may end successfully or in a failed state. Additional options may be attached with a request, which may constrain the time or scope of service discovery. Such kinds of options may stop and fail a discovery process even before the broker of the agent hierarchy has been reached.

Each step for service discovery is processed between nearby agents, while many agents can take part in one service discovery, which may lead to service discovery in a large scope. This principle is the same as that has been applied in service advertisement. Thus service advertisement and discovery in large-scale systems are supported. It is clear that the cost for this is much more complex behaviours for agents. In the next section, a simple example and a formal approach are introduced to give a better understanding of the service discovery processes and their relationship with service advertisement when the system is highly dynamic.

5.4.2 Formal Approach

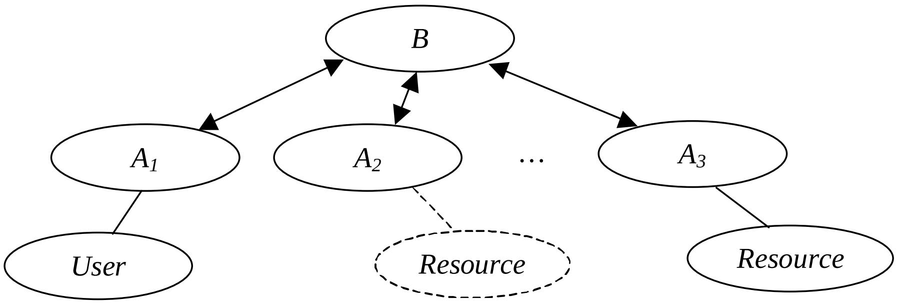


Figure 5.3 An Example System

The example shown in Figure 5.3 is a simple agent system with two levels, one broker with several agents below. Each agent maintains a T_ACT, a L_ACT, and a G_ACT. The broker only has a T_ACT and a L_ACT. Consider a typical process: *User* sends a request, *s*, through agent A_1, and the service can be provided by *Resource*. But the *Resource* just moved from agent A_2 to A_3.

Each T_ACT and L_ACT is maintained by an event-driven data-push method, and the G_ACTs of these agents are updated using a periodic data-pull method. In this situation, when the resource is moved, the related T_ACTs and L_ACTs are all updated immediately, but when the request is sent out, the G_ACTs of these agents have not been updated. How will the service discovery proceed?

The formal representation of the problem is summarised in Table 5.2, which includes the definitions of agents, evaluations, and processes. This is the basis for the rule-based reasoning of system dynamic processes.

Agents	A_i, $(i=1,\ldots\ldots,n)$, one of the agents *s*, a given service request
Evaluations	*t(s)*, evaluation result of *s* in T_ACT *l(s)*, evaluation result of *s* in L_ACT *g(s)*, evaluation result of *s* in G_ACT $t(s), l(s), g(s) \in \{A_i\ (i=1,\ldots\ldots,n), null\}$ *null* means no service information is available for the request *s*
Processes	$A_i(s)$, A_i processes the request *s*

Table 5.2 Formal Representation

We represent the process for an agent to require a service in a logical way. The rules show the routes for a request from the original agent to reach the target agent though the resource can be moved dynamically. Several basic rules are used, which formalise the service discovery process described in the last section.

- **Rule 1**: $A_i(s) \Rightarrow A_i \rightarrow (t(s), l(s), g(s))_{Ai}$

  The service discovery process in an agent is the process of looking up the T_ACT, L_ACT and G_ACT (C_ACT is not used in this case).

- **Rule 2**: $(A_{this}, *, *)_{this} \Rightarrow ServiceFound$

  If an agent is aware that it can provide the required service itself, the service discovery is successful.

- **Rule 3**: *(null, A_{lower}, *)$_{this}$* $\Rightarrow A_{lower}(s)$

  If the required service information cannot be found in the T_ACT but in the L_ACT, the request will be dispatched to the lower agent.

- **Rule 4**: *(null, null, $A_{another}$)$_{this}$* $\Rightarrow A_{another}(s)$

  If the required service information cannot be found in the T_ACT or L_ACT but in the G_ACT, the request will be dispatched to the corresponding agent.

- **Rule 5**: *(null, null, null)$_{this}$* $\Rightarrow A_{upper}(s)$

  If an agent exhausts the ACTs, and does not obtain the required service information, it will submit the request to its upper agent.

- **Rule 6**: *(null, null)$_{broker}$* $\Rightarrow NoService$

  If a broker (head of an agent hierarchy) exhausts the ACTs (G_ACT is not maintained in a broker), the service discovery ends unsuccessful.

These rules can be organised together to reason about the route of the service discovery process in the example system. The equations are shown below. For each step, the evaluation results of all of the ACTs to the request *s* replace the correspondent parts, $(t(s), l(s), g(s))_{Ai}$, in the process automatically. The number at the end of each line indicates the rule used for the transformation.

$$A_1(s) \Rightarrow A_1 \rightarrow (null, null, A_2)_{A_1} \qquad (1)$$
$$\Rightarrow A_1 \rightarrow A_2(s) \qquad (4)$$
$$\Rightarrow A_1 \rightarrow A_2 \rightarrow (null, null, null)_{A_2} \qquad (1)$$
$$\Rightarrow A_1 \rightarrow A_2 \rightarrow B(s) \qquad (5)$$
$$\Rightarrow A_1 \rightarrow A_2 \rightarrow B \rightarrow (null, A_3, *)_B \qquad (1)$$
$$\Rightarrow A_1 \rightarrow A_2 \rightarrow B \rightarrow A_3(s) \qquad (3)$$
$$\Rightarrow A_1 \rightarrow A_2 \rightarrow B \rightarrow A_3 \rightarrow (A_3, *, *)_{A_3} \qquad (1)$$
$$\Rightarrow A_1 \rightarrow A_2 \rightarrow B \rightarrow A_3 \rightarrow ServiceFound \qquad (2)$$

Three connections are needed for the A_1 to find the required service in A_3. In the G_ACT of A_1 the service is still recorded to be within the capability of A_2. A_2 still has to take part in the routing process. The routing process can be simplified if A_2 can cache this routing result or the G_ACT of A_1 can be updated some time later.

The system can have more than two levels and the services may be changed many times. The system behaviours for service discovery may become much more complex. Modelling and simulation tools can be developed to estimate the system performance, as introduced in the following sections.

5.5 Performance Metrics

Unlike other service discovery infrastructures that focus on data models and communication protocols, the A4 methodology focuses on performance issues that arise from system dynamics. Two extreme situations can be considered:

- No service advertisement - results in complex service discovery. In this situation no ACTs are maintained in the agents. Each agent has no knowledge of the services offered by other agents. When a service is requested, a service discovery process is required which may be complex and may traverse a large number of agents in the system.
- Full service advertisement - requires no service discovery. In this situation, each agent advertises as much as possible to the other agents. Hence each agent has nearly complete knowledge of the available services in the system and no discovery process is required. When a request is made, the service is found in any agents ACT.

Different systems can use different optimisation to achieve high performance. For example in static systems, where the frequency of change in the service information is far less than the frequency of service requests, more service advertisement can achieve high performance service discovery. In extremely dynamic systems, where the frequency of change in the service information is far greater than the request frequency, less service advertisement can achieve high performance. Most practical systems will have characteristics in-between these two extremes.

There are different kinds of performance criteria that can be used to describe the service discovery performance part of the system. What is considered as high performance depends on the system requirements. However, there are some common characteristics of the system that are usually a concern to the system developer. These include discovery speed, system efficiency, load balancing, and success rate, which will be discussed below.

5.5.1 Discovery Speed

Each request from an agent can pass one or more agents in order to find a target agent that can provide the required service. Fewer connections have a quick discovery process, and higher system performance. In the whole system, there may be simultaneous service requests. The average service discovery speed, v is defined as:

$$v = \frac{r}{d}$$

where r is the total number of requests during a certain period, and d is the total number of connections made for the discovery.

The performance of the discovery process is mainly based on the number of routing connections. The communication time for each connection is not considered here to simplify the performance modelling and simulation of the agent system.

5.5.2 System Efficiency

The cost for the service discovery also includes connections made for service advertisement and data maintenance. Service advertisement may add additional workload to the system. For each request to find a corresponding service, the total number of connections, c, between agents includes those for the discovery processes, d, and also those for the advertising processes, a.

$$c = d + a$$

The efficiency of the system can be considered as the ratio of the total number of requests, *r*, during a certain period, to the total number of connections *c*.

$$e = \frac{r}{c}$$

5.5.3 Load Balancing

In some of the systems when the system resources are critical, load balancing may be an important issue. In the A4 methodology, no agents are used only for service discovery. There is no reason to have any agent with a higher discovery workload than any other. For a system with *n* agents, the workload, w_k, of each agent can be described as

$$w_k = o_k + i_k \qquad (k = 1......n)$$

where o_k and i_k are the outgoing and incoming connection times. We can use the mean square deviation of the w_k to describe the load balancing level of the system, *b*:

$$b = \sqrt{\frac{\Sigma_k \left(w_k - \overline{w}\right)^2}{n}} \qquad \text{where} \quad \overline{w} = \frac{\Sigma_k w_k}{n}$$

5.5.4 Success Rate

In some situations the discovery model cannot guarantee to find the target service (that may actually exist in the system). However, in a general system a reasonable service discovery success rate should always be achieved. The success rate, *f*, describes successful service discovery:

$$f = \frac{r_f}{r} \times 100\%$$

Most of the time, these service discovery metrics may conflict, that is not all metrics can be high at the same time. For example, a quick discovery speed does not mean high efficiency, as sometimes quick discovery may be achieved through the high workload encountered in service advertisement and data maintenance, leading to low system efficiency. It is necessary to find the critical factors of a practical system, and then to use the different agent configurations to reach high performance.

5.6 A4 Simulator

Performance evaluation of service discovery in a large-scale multi-agent system is a difficult task. Different configurations of agent behaviours on service advertisement and discovery can make the overall system behaviours very complex. In this section, a modelling and simulation environment, the A4 simulator, is introduced.

The A4 simulator has as input all of performance related information of the agent system, it composes them into a performance model, simulates the service advertisement and discovery processes step by step, and finally outputs all of the statistical data on the four performance metrics described above.

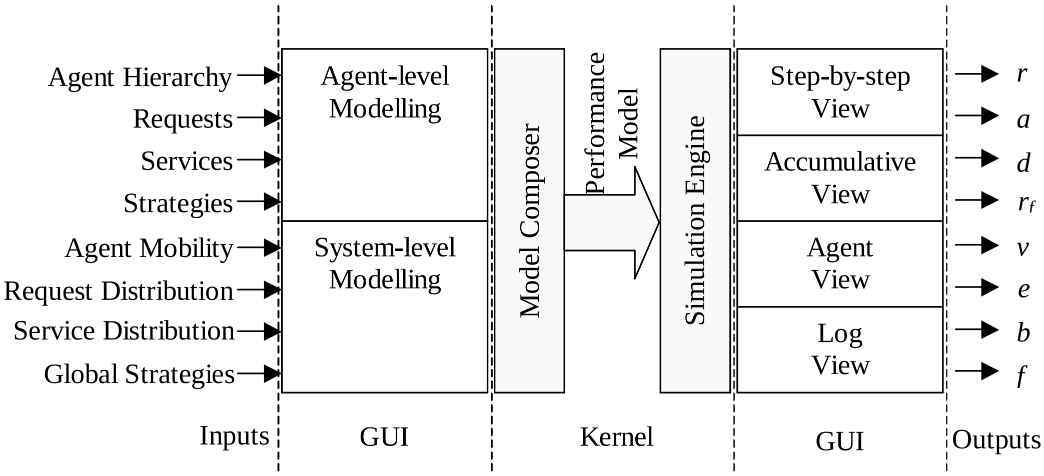


Figure 5.4 A4 Simulator

The main structure of the A4 simulator is illustrated in Figure 5.4, which includes a kernel and GUIs. The kernel part of the simulator performs the modelling and simulation functions, while users can input related information and get simulation outputs from the GUIs.

5.6.1 Inputs/Outputs

There are four kinds of information that affect the system performance and must be input into the performance model. These include: the agent hierarchy, the services, the requests, and the strategies for service advertisement and discovery. The A4 simulator supports the modelling activity at both the agent level and the system level. The only components that exist in the model are agents, so agent-level modelling can be used to define all the model attributes for the simulation. However, system-level modelling is also necessary to input information on agent mobility, service and request distribution, and so on. These will be discussed in detail below.

- Agent hierarchy. When a new agent is added into the model, its upper agent should be defined. The upper agent is also configured to add a new lower agent. The information is used to organise agents into a hierarchy in the system model. No cycles are permitted in the hierarchy, which may cause deadlock during the service discovery process.
- Requests. Each agent is configured to send different requests periodically. A request item may include several parts of information: the required service name, the relative required performance value, the sending frequency, and the discovery scope.
- Services. Each agent is also configured to provide many services, whose performance may vary over time. A service item may include several parts of information, the service name, the relative performance value, the performance changing frequency, service available time, and service advertisement scope. The usage of these attributes will be introduced in the simulator kernel section below.
- Strategies. Different strategies are defined in each agent to control its behaviours on service advertisement and discovery. These strategies have been discussed in detail in Section 5.3 and 5.4 respectively.

- Agent mobility. The agent mobility can be defined at the system level only. An agent mobility item may include information on: the agent ID, the new agent ID after the movement, the upper agent ID of the new agent, and the step number when the movement will happen during the simulation.
- Request distribution. System-level request definitions can ease the modelling process. The same request item does not need to be defined in different agents one by one. The A4 simulator provides a convenient way to distribute a request definition to different agents once it is defined at the system level.
- Service distribution. The same service with the same attributes can also be provided by different agents. System-level service definitions allow many agents to be configured with the same service at the time.
- Global strategies. A system-level strategy definition can affect all of the agents in the model and ease the modelling process. Both global strategies and individual strategies can be defined in each agent. However, agent-level strategy definitions have a priority over the system-level ones.

The information above is input into the simulator. The outputs of the simulator are all of the simulation results on four performance metrics. All of the details on service advertisement and discovery are also recorded in a simulation log file for further reference. The use of input information to produce outputs during the modelling and simulation processes within the simulator kernel is introduced below.

5.6.2 Simulator Kernel

The kernel of the simulator is composed of a model composer and a simulation engine. The kernel will perform the main modelling and simulation functions and transform the raw simulation data to statistical results to support the four performance metrics.

The model composer organises the input information into a performance model before the simulation process begins. During this phase, the system-level information is transferred into an agent-level representation as much as possible. For example, system-level requests and services will be used to configure a certain percentage of agents. The global strategies are used to define the strategies of each agent, except for agents that have already been defined with agent-level strategies. After these, a performance model is composed and the simulator is ready for evaluation. The information on agent movement can only be stored at the system level and will not be used to configure any agent in the system.

The simulation engine will start a simulation process once a performance model and a total number of simulation steps are defined. The whole process is illustrated in Figure 5.5, which is divided into seven phases, five of which are within the main simulation loop.

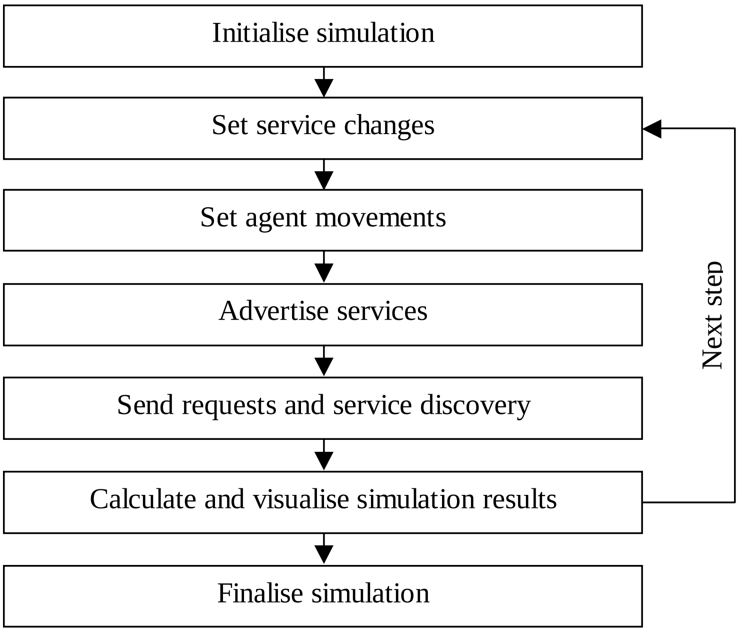


Figure 5.5 Simulation Process of A4 Simulator

- Initialise simulation. Once a simulation process is started, the A4 simulator will set up an environment for simulating service advertisement and discovery. All of the GUIs for performance modelling are locked. The performance model cannot be modified during the simulation. A copy of the model is also made to prevent data loss due to the simulation being irregularly interrupted. The simulation results are also initialised for recording the outputs.

- Set service changes. This is performed at the beginning of each simulation step. The availability and performance of a service may change at each step. The service available time in each service item records the step number when the corresponding service is available. After that, the service will be deleted in all ACTs of all agents in the model. There is also the frequency of change in performance of each service. The performance of each service may or may not be changed at each step according to this frequency.

- Set agent movements. Each agent mobility item contains a step number when a movement will happen during the simulation. An agent movement indicates not only the change of the agent hierarchy, but also the change of related services. Additional service advertisement occurs when an agent is moved, for example, old service information is announced for deletion, and new service information should be advertised along the new agent hierarchy. An agent is moved while its upper agent may or may not be changed, which leads to different situation with different service advertisement workload.

- Advertise services. Both event-driven and periodic service advertisement are considered during this phase. Each agent acts on its ACTs according to its strategy configurations. Each connection between agents for service advertisement will be recorded in the simulation log file and will effect corresponding simulation results.

- Send requests and service discovery. A request is decided to be sent according to its frequency. Each agent that receives the request will look up its ACTs in turn according to its strategy configuration for service discovery. Every detail of a service discovery process is recorded in the log file and related simulation results, such as agent connection times, are recorded.

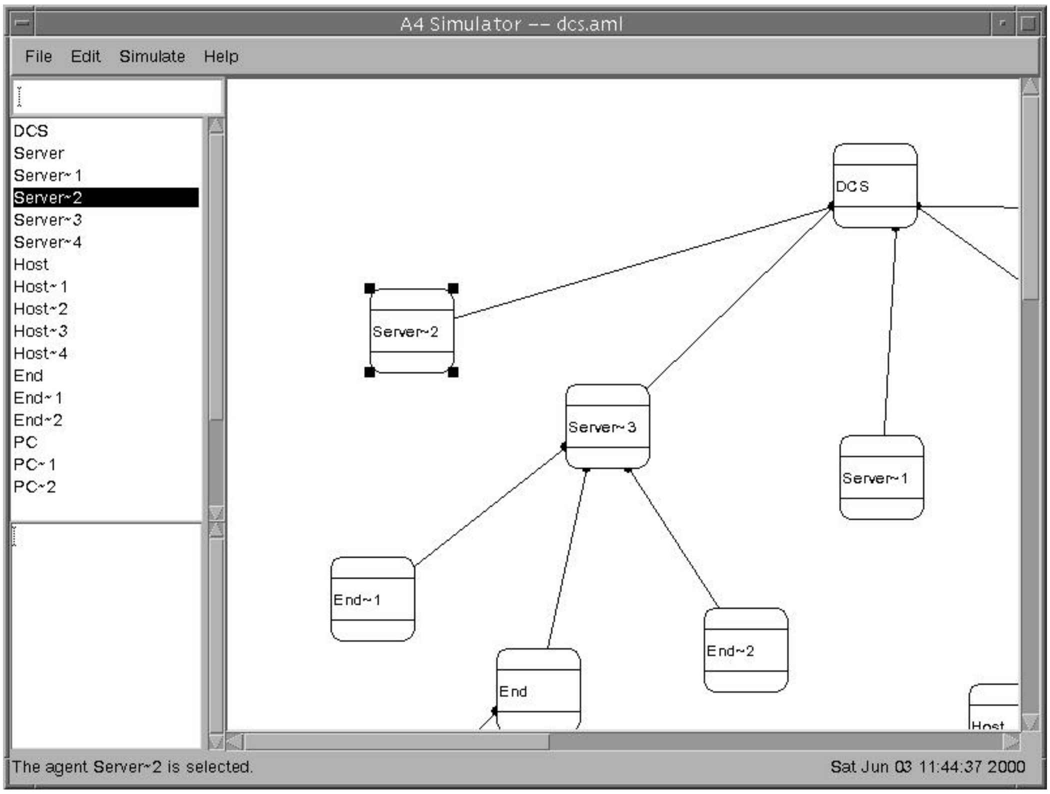
- Calculate and visualise simulation results. At the end of each simulation step, the raw simulation data should be summarised, and corresponding statistical results on the performance metrics calculated. These results are shown on the simulator GUI dynamically to give the user a view of what is going on during the simulation.

- Finalise simulation. After all simulation steps are completed the simulator returns back to the modelling mode. All the modelling GUIs are unlocked. The performance model is retrieved from the original copy. The GUIs for visualising the simulation results will not be refreshed until the next simulation begins, and can thus be used for further analysis.

The A4 simulator also supports the evaluation of multiple models simultaneously. The user can use different configurations in different models, simulate them, and compare the results.

5.6.3 User Interfaces

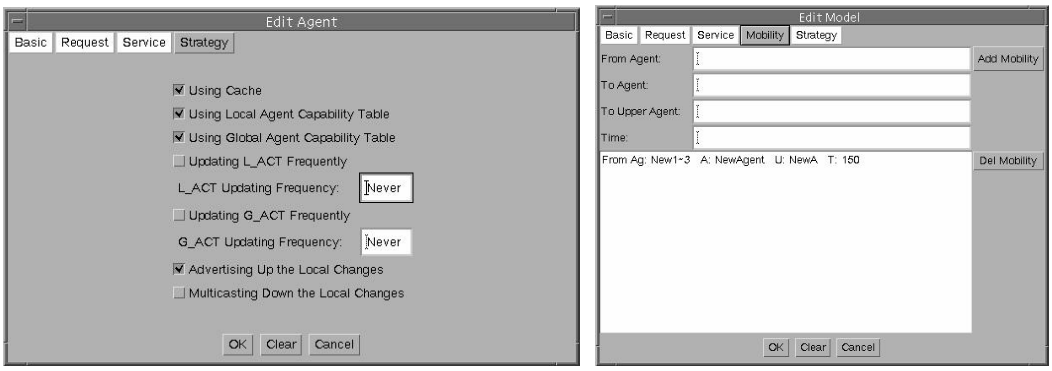
The A4 simulator is implemented using Java. It provides graphical user interfaces for the modelling and simulation respectively.



The agent Server~2 is selected.

Sat Jun 03 11:44:37 2000

(a) Main window



Edit Agent

Basic | Request | Service | Strategy

☑ Using Cache
☑ Using Local Agent Capability Table
☑ Using Global Agent Capability Table
☐ Updating L_ACT Frequently
L_ACT Updating Frequency: Never
☐ Updating G_ACT Frequently
G_ACT Updating Frequency: Never
☑ Advertising Up the Local Changes
☐ Multicasting Down the Local Changes

OK | Clear | Cancel

Edit Model

Basic | Request | Service | Mobility | Strategy

From Agent:
To Agent:
To Upper Agent:
Time:

From Ag: New1~3 A: NewAgent U: NewA T: 150

Add Mobility | Del Mobility

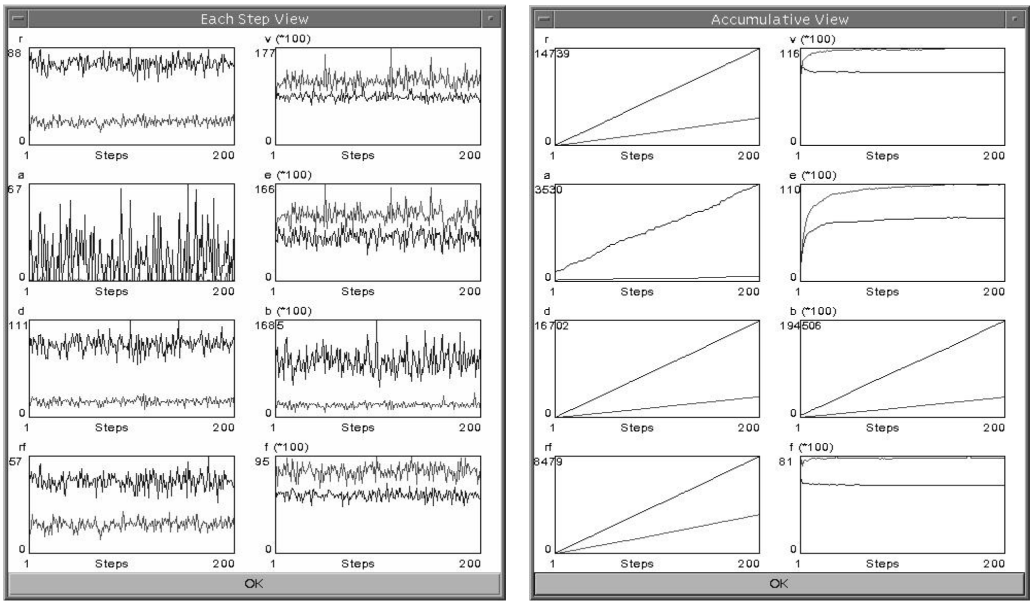
OK | Clear | Cancel

(b) Agent-level modelling (c) System-level modelling

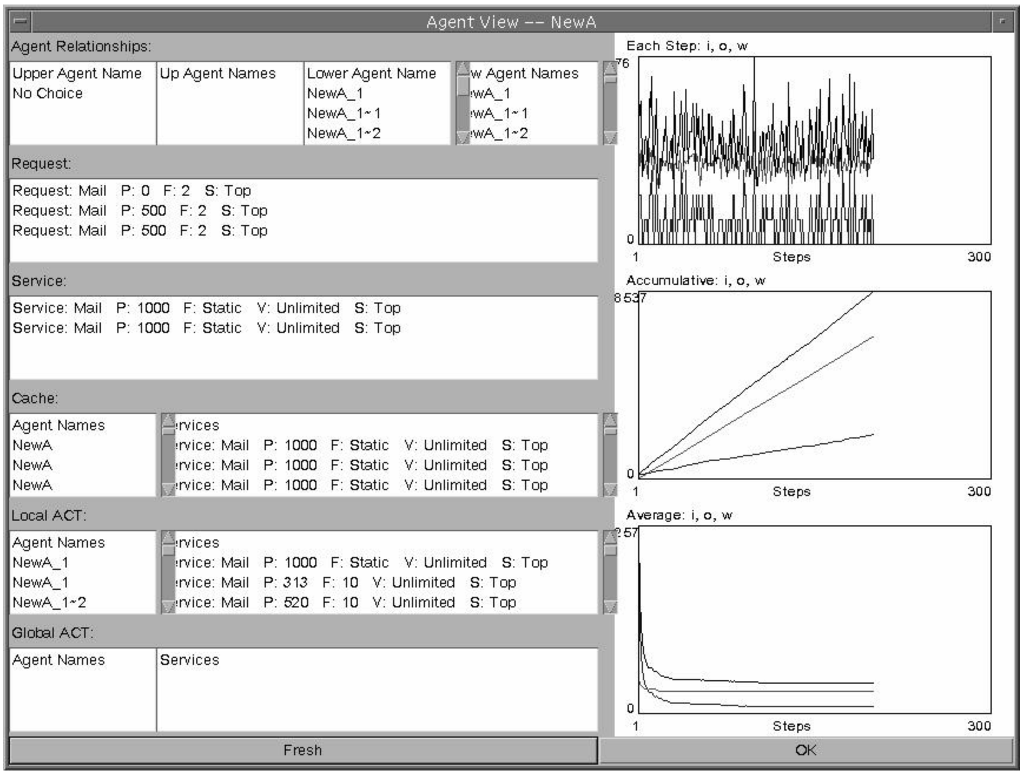
Figure 5.6 A4 Simulator GUIs for Modelling

The user can add, edit and delete agents from the model via the main GUI window shown in Figure 5.6(a). In the left column of the main window, all of the agents are listed. A brief description of the selected agent is also shown below the agent list. The text field above the agent list can be used to search an agent by its name. The model can also be saved and reloaded for reuse later. The windows shown in Figure 5.6(b) and Figure 5.6(c) can be used for agent-level and system-level modelling respectively.

Some other GUIs in the A4 simulator are used to visualise simulation results to the user, which are shown below in Figure 5.7.



(a) Step-by step view (b) Accumulative view



Agent Relationships:

Upper Agent Name	Up Agent Names	Lower Agent Name	w Agent Names
No Choice		NewA_1	wA_1
		NewA_1~1	wA_1~1
		NewA_1~2	wA_1~2

Request:

Request: Mail P: 0 F: 2 S: Top
Request: Mail P: 500 F: 2 S: Top
Request: Mail P: 500 F: 2 S: Top

Service:

Service: Mail P: 1000 F: Static V: Unlimited S: Top
Service: Mail P: 1000 F: Static V: Unlimited S: Top

Cache:

Agent Names	rvices
NewA	rvice: Mail P: 1000 F: Static V: Unlimited S: Top
NewA	rvice: Mail P: 1000 F: Static V: Unlimited S: Top
NewA	rvice: Mail P: 1000 F: Static V: Unlimited S: Top

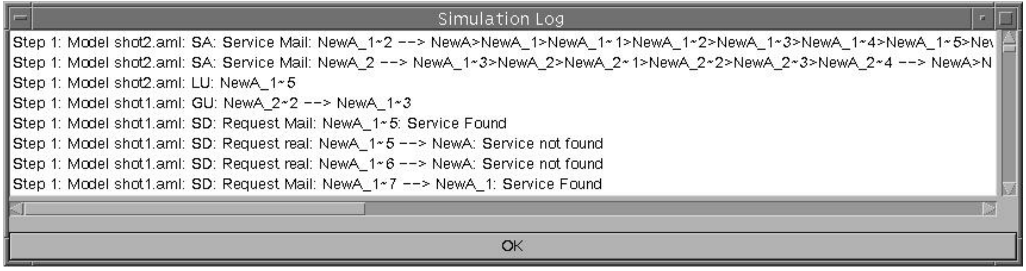
Local ACT:

Agent Names	rvices
NewA_1	rvice: Mail P: 1000 F: Static V: Unlimited S: Top
NewA_1	rvice: Mail P: 313 F: 10 V: Unlimited S: Top
NewA_1~2	rvice: Mail P: 520 F: 10 V: Unlimited S: Top

Global ACT:

Agent Names	Services

(c) Agent view



```
Step 1: Model shot2.aml: SA: Service Mail: NewA_1~2 --> NewA>NewA_1>NewA_1~1>NewA_1~2>NewA_1~3>NewA_1~4>NewA_1~5>Nev
Step 1: Model shot2.aml: SA: Service Mail: NewA_2 --> NewA_1~3>NewA_2>NewA_2~1>NewA_2~2>NewA_2~3>NewA_2~4 --> NewA>N
Step 1: Model shot2.aml: LU: NewA_1~5
Step 1: Model shot1.aml: GU: NewA_2~2 --> NewA_1~3
Step 1: Model shot1.aml: SD: Request Mail: NewA_1~5: Service Found
Step 1: Model shot1.aml: SD: Request real: NewA_1~5 --> NewA: Service not found
Step 1: Model shot1.aml: SD: Request real: NewA_1~6 --> NewA: Service not found
Step 1: Model shot1.aml: SD: Request Mail: NewA_1~7 --> NewA_1: Service Found
```

(d) Log view

Figure 5.7 A4 Simulator GUIs for Simulation

During each step in the simulation the results will be updated in each of the GUIs. The simulator can provide multiple views of the simulation data, which are all updated in real time. In the step-by-step view of the Figure 5.7(a), the simulation data, r, a, d, r_f, and the statistic data, v, e, b, f, in each step are shown. In the accumulative view shown in Figure 5.7(b), the statistical data on the accumulative steps are shown. In the agent view shown in Figure 5.7(c), the user can view the contents of a selected agent, its operation at each step, accumulative and average views of the data o_k, i_k, and w_k. In Figure 5.7(d), the log view shows the simulation log file, which records the details of all service advertisement and discovery processes during simulation.

5.6.4 Main Features

The A4 simulator is developed to provide quantitative information of the performance of service advertisement and discovery in large-scale multi-agent systems with highly dynamic behaviours using the A4 methodology. The main feature of the A4 simulator can be summarised as follows:

- Support for all of the performance metrics and strategy configurations described in the A4 methodology;
- Support two levels of system modelling for easy and convenient performance modelling of multi-agent systems;
- Support modelling of agent mobility and simulation of additional service advertisement processes;
- Support multi-view and real-time display of simulation results;
- Support simultaneous simulation of multiple models and comparison of results;
- Support simulation log management.

The use of the A4 simulator for a performance study is introduced in the next section through a case study, and simulation results are included to show the impact of agent mobility on the system service discovery performance. Meanwhile, the A4 simulator kernel can also be used in practical multi-agent systems to analyse and optimise system service discovery performance on-line, which will be introduced in Chapter 7.

5.7 A Case Study

In Section 5.4.2, a simple example with a formal representation was given. A resource in the system was moved, which results in more workload for service discovery. In this section, the A4 simulator is used to study the impact of agent mobility on system service discovery performance using a much more complex example.

5.7.1 Performance Model

A simple multi-agent system model is shown in Figure 5.8, containing 26 agents. The whole system is configured to have only one service named *Print*. The agent that can provide the service is *Printer* now connected to *till* and later, during the simulation, is moved to connect to *sun* with a new identity *NewPrinter* (this is not shown in Figure 5.8). All the other agents may or may not request the *Print* service with a different frequency (Note that the details of requests are not given below).

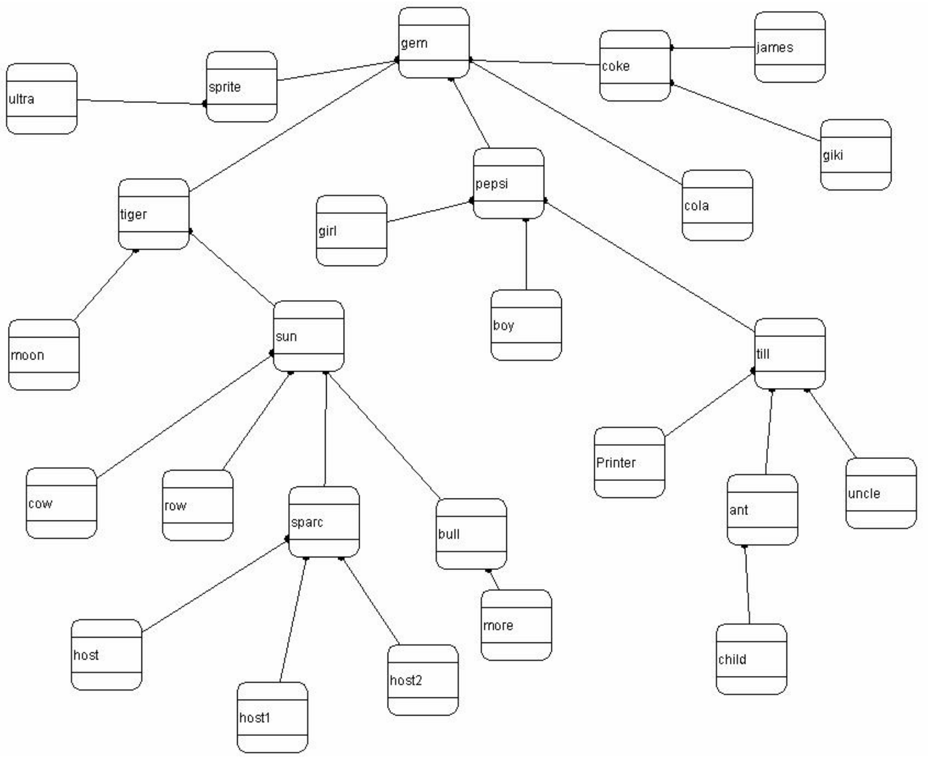
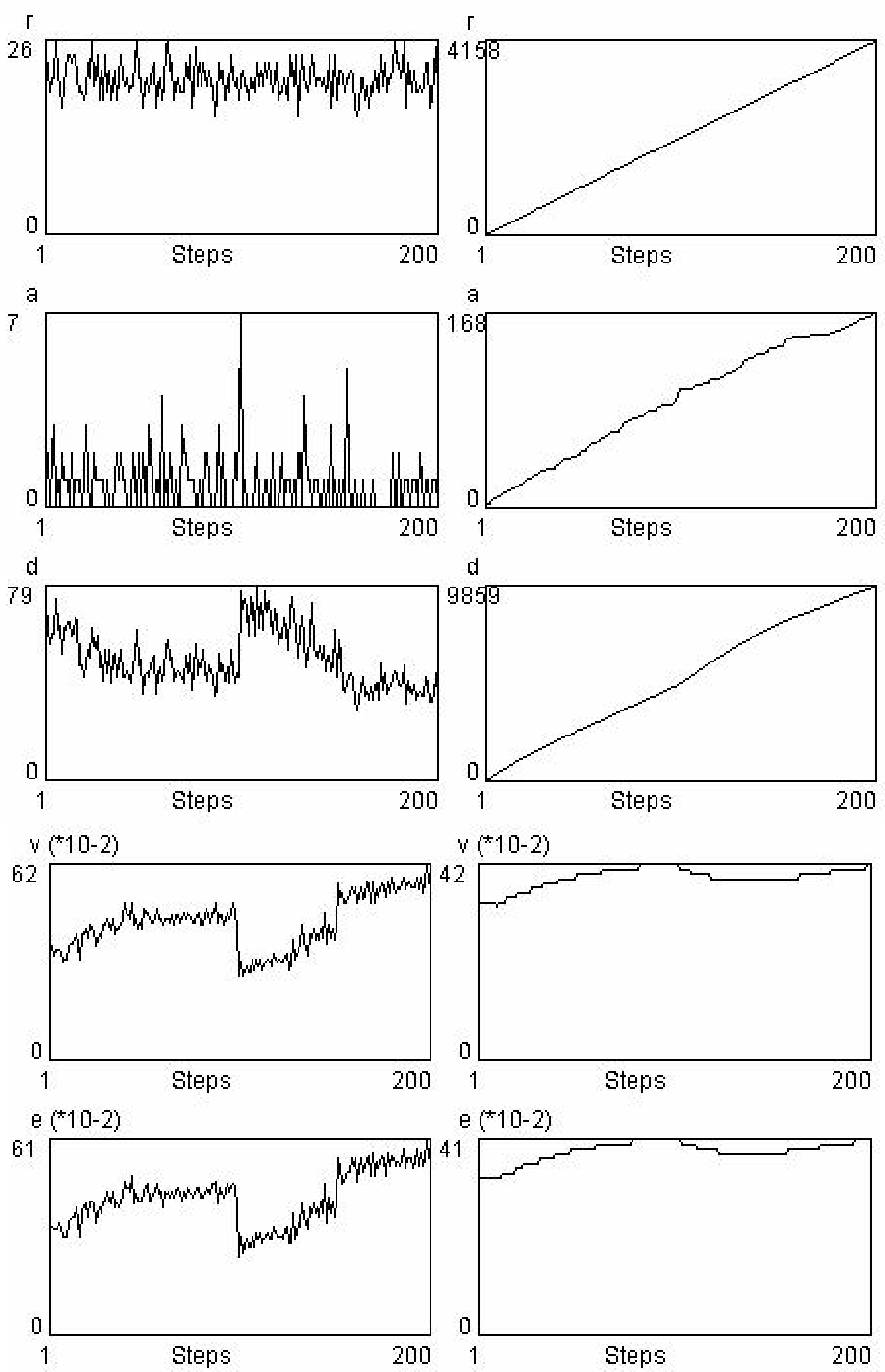


Figure 5.8 Example Model: Agent Hierarchy

This experiment is used to show the impact of agent mobility on the service discovery performance. Strategies are only defined at the system level, which means that all of the agents in the model must use the same strategies for service advertisement and discovery. The T_ACT, L_ACT and G_ACT are used in each agent. T_ACTs and L_ACTs are maintained by event-driven service advertisement. G_ACTs are updated once every 30 steps using a periodical data-pull. The agent movement mentioned above takes place at the 100th simulation step.

5.7.2 Simulation Results

Figure 5.9 shows the simulation results for 200 steps. A step can be designed as an arbitrary number of seconds. The curves for discovery speed (*v*), and the system efficiency (*e*) in the step-by-step view show the effect of the agent mobility most clearly.



(a) Step-by-step View (b) Accumulative View

Figure 5.9 Simulation Results

We assume that the load balancing and discovery success rate are not critical in this study. Attention is given to the discovery speed and the system efficiency. The whole process can be divided into five phases, which are explained in detail below.

- Learning phase. In the first 40 steps, the G_ACTs of the agents are updated gradually, so the discovery speed and system efficiency increase. This can be viewed as an agent learning process.
- Stable phase. After about 40 steps, the curves are flat at a higher level. All G_ACTs of the agents have been updated and there are no service changes, so the system runs in a steady state mode with high service discovery speed and system efficiency.
- Agent mobility. The defined agent mobility happens at the 100th simulation step. When the agent moves it must advertise to delete its service information from the old agent hierarchy and to add the new service information to the new agent hierarchy. This causes an increase of the connections for service advertisements (*a*). The service information in all the agents becomes out-of-date, which results in more workload for the service discovery (*d*). So the average service discovery speed (*v*) and system efficiency (*e*) decrease suddenly.
- New learning phase. This phase is the same as the previous learning phase. The agents learn about the new identity of the service *Print* gradually via the G_ACT updating.
- New stable phase. The agent mobility finally results in a stable state mode with higher performance. This is because *sun* is the coordinator of a larger sub-hierarchy than *till* is. When the service is moved, more requests become local instead of remote, which reduces the discovery workload of the system.

This is a small example model with only one agent movement. The system model is not a large-scale one and the service in the system is static during most of the simulation time. However, this simple case study gives an intuitive impression that system dynamics has a great impact on the service discovery performance.

The A4 simulator enables such kinds of problems to be investigated quantitatively.

5.8 A4 as a Global Framework

The aim of this work is the development of a grid resource management system. In Section 4.4, we have introduced PACE functions that can be used for local resource management in a grid environment. In this section, we discuss that the A4 methodology can be applied as a global framework to implement meta-level grid resource management.

Agents are the main abstractions in the A4 methodology. An agent can be used as a representative of a local high performance resource in a grid environment. The high performance computing capability that a local resource can provide is modelled as a service. Each agent is a service provider of high performance computing.

Each agent can also be equipped with PACE performance prediction capabilities in its local resource management for scheduling parallel applications to available local resources. PACE functions are also used in the coordination layer of agents to provide QoS support for service discovery.

Each agent is responsible for local resource monitoring, and corresponding service information is collected and stored in the T_ACT. An agent is also responsible for advertising the service through the agent hierarchy, according to different strategy configurations.

Grid users can send application execution requests to the grid environment, which can be received by a nearby agent. Agents can cooperate with each other and perform service discovery functions to find an available service for the requests.

When a target agent is found that can provide the requested service, the user can contact the agent directly for application execution. Hence A4 can provide a global framework and be coupled with PACE functions to implement grid resource management. An initial implementation of an agent-based resource management system for grid computing, ARMS, will be described in detail in the next chapter.

Chapter 6

ARMS:
AGENT-BASED RESOURCE MANAGEMENT SYSTEM FOR GRID COMPUTING

Resource management in the grid computing environments will rely on accurate application performance prediction capabilities, as discussed in Chapter 4. An agent-based methodology is also introduced in the last chapter to address the challenges of scalability and adaptability. In this chapter, an initial implementation of an agent-based resource management system for grid computing, ARMS, is presented [Cao2001d], using a hierarchy of homogenous agents [Cao2001b] coupled with the prediction capabilities of the performance evaluation toolkit, PACE.

6.1 ARMS in Context

The relationship between ARMS and other concepts mentioned in this thesis is shown in Figure 6.1. ARMS is a system, which builds a bridge between grid users and resources to schedule applications to utilise the available grid resources.

PACE is used to provide quantitative data concerning the performance of sophisticated applications running on local high performance resources. PACE application tools (AT) are provided to grid users. A request to execute an application by a user must be attached with a corresponding application model developed using the AT. Meanwhile, PACE resource tools (RT) are embedded in each grid resource to provide a corresponding resource model, which is an important part of the service information of the resource. The PACE evaluation engine (EE) is used in each agent in the ARMS for performance evaluation given both the application and resource models.

At a metacomputing level, the A4 methodology is used for grid resource management. Agents cooperate with each other and perform service advertisement and discovery functions to schedule applications that need to utilise the available resources. The behaviours of agents can be configured with different strategies and steered with different policies to improve the system performance.

A performance monitor and advisor, PMA, is a special agent existing in the agent system of ARMS. The main part in the PMA is the A4 simulator kernel. PMA monitors the state of each agent, configures each agent with modelling and simulation results, and steers the agent behaviours to implement the resource management more efficiently. PMA will be introduced in detail in the next chapter.

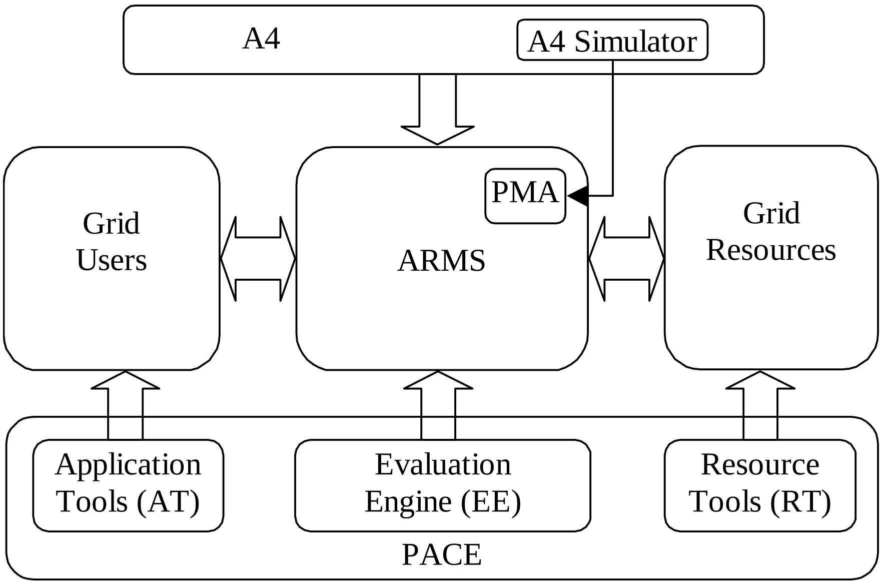


Figure 6.1 ARMS in Context

6.2 ARMS Architecture

ARMS is an agent-based grid resource management system. An overview of the ARMS architecture is illustrated in Figure 6.2. The main components in the architecture include: grid users, grid resources, ARMS agents, and the ARMS PMA. These will be discussed respectively in the following sections.

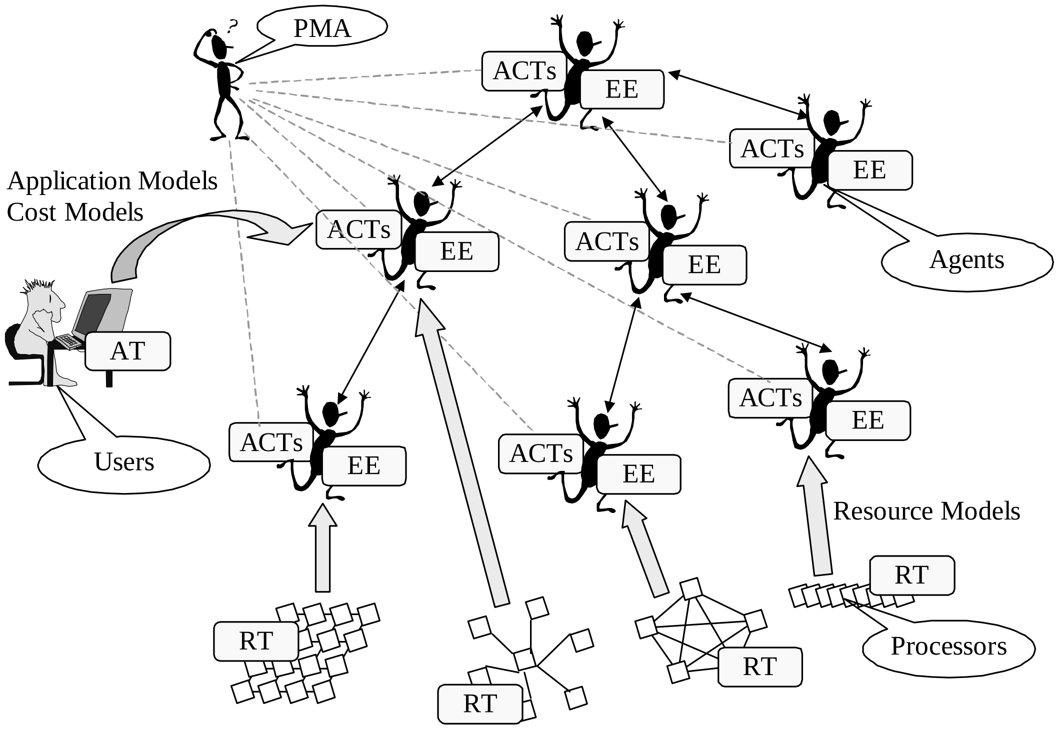


Figure 6.2 ARMS Architecture

6.2.1 Grid Users

There are different kinds of users of a grid computing environment. Grid developers are responsible for implementing basic grid services. ARMS provides grid resource management, which is a part of these services.

The developers of the tools, compilers, libraries, and so on implement the programming models and services used by application developers. MPI and PVM are included in these kinds of tools. Grid service and tool developers are a very small group of grid users, which are not of concern in the context of this thesis.

Application developers comprise those who construct grid-enabled applications using grid tools. There are different kinds of grid applications: distributed supercomputing, high throughput, on demand, data intensive, and collaborative applications. The applications mentioned in this work mainly refer to scientific supercomputing applications, which are very large problems needing lot of CPU, memory, etc, especially those written in MPI and PVM.

Most grid users, like most users of computers or networks today, will not write programs. Instead, these end users will use grid-enabled applications that make use of grid resources and services. In some situations, application developers are also the end users of the applications they develop. The grid users in Figure 6.2 and mentioned in the following sections are considered to be scientists, who develop scientific supercomputing applications and use them to solve large problems in the grid environment.

As shown in Figure 6.2, grid user side software includes the PACE application tools. When a parallel application is developed, the corresponding application model should also be produced using PACE tools. As described earlier, performance modelling using PACE is an easy process that can be used by non-professional performance engineers. Each request to execute an application that is sent to a grid environment should be attached with a corresponding PACE application model.

Another component included in a grid request is the cost model, which describes all information on a user's requirements about the application execution, for example, the deadline for the application execution to be finished. Though there can be many metrics for application execution, we focus on application execution time only here.

6.2.2 Grid Resources

A grid resource can provide high performance computing capabilities for grid users. A resource can include Massive Parallel Processors (MPP), or a cluster of many workstations, or even PCs. A grid resource can be considered as a service provider of high performance computing capabilities.

PACE resource tools can be used in each grid resource to provide the model of the resource. The computational and communication benchmark programs can be controlled to execute on the resource to produce performance data for the models dynamically. The PACE resource model is a part of service information of the resource, which will be advertised across the agent hierarchy.

6.2.3 ARMS Agents

Agents are the main components in ARMS. Each agent is a representative of a grid resource at the meta-level of resource management. As introduced in the A4 methodology, agents are organised into a hierarchy. The hierarchy of homogenous agents provides a meta-level view of the grid resources. The service information of each grid resource can be advertised in the hierarchy (both upwards and downwards). Agents can also cooperate with each other to discover an available resource for a request of application execution.

Two important components within each agent are also shown in Figure 6.2. As mentioned in the A4 methodology, each agent has ACTs to record service information of other agents. The service information contains all performance related information of a grid resource, which can be used to estimate its performance.

The PACE evaluation engine is also integrated into each agent. Its performance prediction capabilities can be used for local resource management to schedule parallel applications to available local processors. The PACE evaluation engine is also used in the coordination layer of each agent to provide QoS support for service discovery.

Each agent receives requests from grid users or other agents in the system. How does an agent process to make service discovery decisions using the PACE evaluation engine? How does an agent collect service information from its local resource management? Such kinds of questions will be answered in Section 6.3, where the structure and functions of each agent is described in detail.

6.2.4 ARMS Performance Monitor and Advisor

A special agent is introduced into the ARMS agent system. It is an agent acting as a performance monitor and advisor (PMA). It contacts each agent in the hierarchy as shown in Figure 6.2. The PMA uses the kernel of the A4 simulator, which aims to improve ARMS service discovery performance. We will introduce the structure and functions of the PMA separately in Chapter 7.

ARMS is implemented using the A4 methodology coupled with PACE functions. All functions developed in ARMS correspond to elements of the A4 methodology. However, the detailed implementation of each agent need also be described in the next section.

6.3 ARMS Agent Structure

The agent structure in ARMS is shown in Figure 6.3, which corresponds to the general A4 agent structure shown schematically in Figure 5.2. Each layer has several modules, which cooperate with each other to perform service advertisement and discovery functions.

The communication layer of each agent performs communication functions and acts as an interface to the external environment. From the communication module, an agent can receive both service advertisement and discovery messages. It handles the contents in the message and submits them to corresponding modules in the coordination layer of the agent. For example, an advertisement message from other agents will be directly sent to the ACT manager in the agent coordination layer. The communication module is also responsible for sending out messages for service advertisement or discovery to other agents.

There are four components in the coordination layer of an agent: ACT manager, PACE evaluation engine, scheduler, and matchmaker. They work together to make decisions on how an agent should act on the received messages from the communication layer. For example, the final response to a service discovery message includes: application execution on the local resource or dispatching the request to another agent.

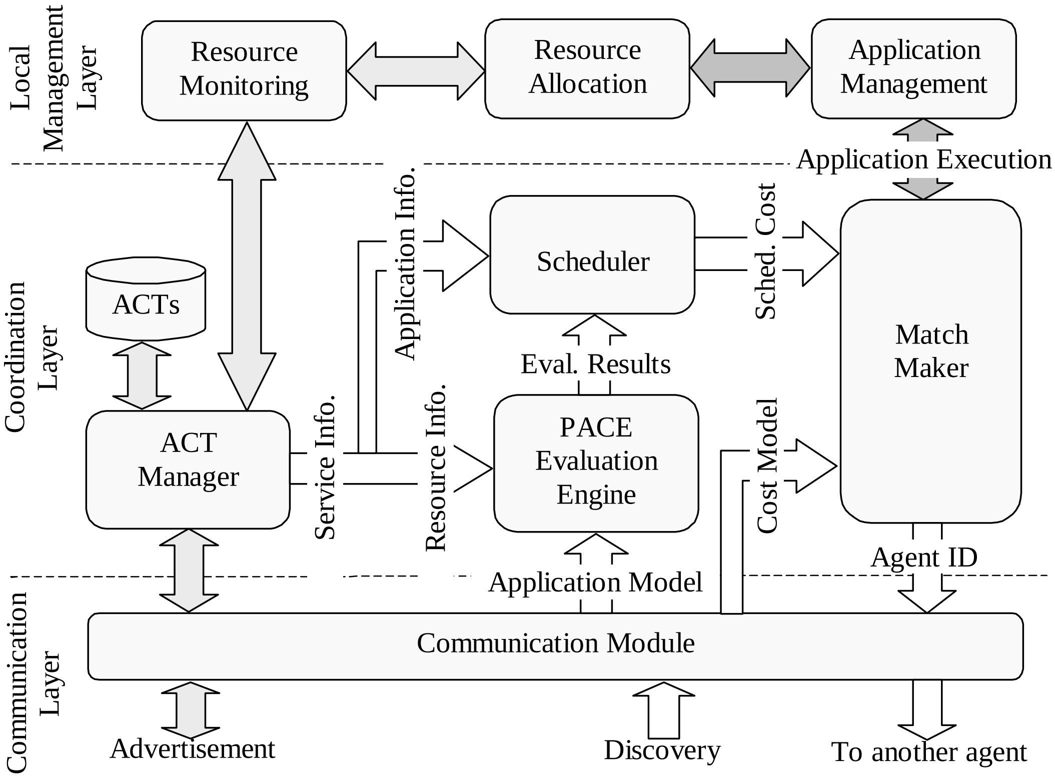


Figure 6.3 ARMS Agent Structure

The main functions for local resource management in an agent include application management, resource allocation, and resource monitoring. An application execution command is sent from the coordination layer to local management in an agent, which includes the scheduling information for an application, such as its starting time, and allocated processor IDs. Application management is responsible for managing the queuing applications that have been scheduled to be executed on local resources. When the starting time of an application arrives, it will be dispatched to the resource allocation. Resource allocation has wrappers with different application execution environments like MPI and PVM, and actually implements application execution on scheduled processors. Another important module for local resource management in an agent is resource monitoring. It is responsible to control PACE benchmark programs to be executed on the local resource and construct the corresponding resource models dynamically. The resource monitoring is also responsible for contacting the application management and resource allocation modules for other resource and application information. It will organise all of the collected information about the local resource into service information provided by the local resource and report it to the T_ACT in the coordination layer of the agent.

We describe the agent functions above. As mention before, our work focuses on the implementation of functions for the agent coordination layer. The four main components will be introduced in detail below, and there will be no further introduction to other modules for communication and local management in an agent.

6.3.1 ACT Manager

The ACT manager controls the agent access to the ACT database, where service information of grid resources are recorded. As mentioned in Section 5.3.1, an ACT item contains three parts: agent ID, service information, and additional options. The specific contents of service information in ARMS are shown in Figure 6.4 and explained below.

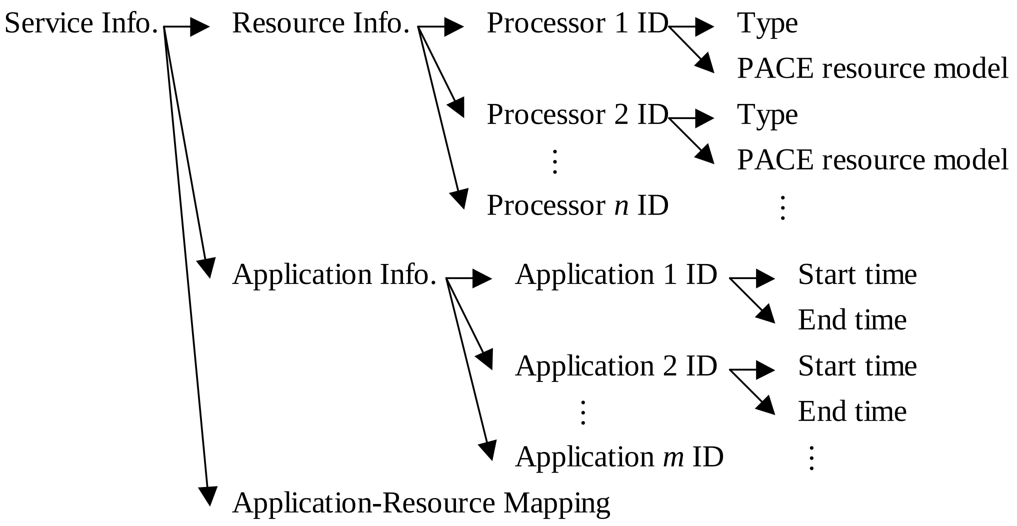


Figure 6.4 Service Information in ARMS

Service information of a grid resource should include all of the information about a resource that has an impact on the performance of a resource and can be used to evaluate its performance. Service information is basically composed of resource information, application information, and the mapping between the applications and the resources.

Consider a grid resource with n processors. Each processor P_i has its own type ty_i, such as Sun Ultra1 and SGI Origin2000. A PACE resource model can be used to describe all the performance information of a processor. PACE resource models of some typical processors can also be pre-installed into the evaluation engine in each agent, instead of running benchmark programs on resources dynamically to produce resource models. In this case, resource models cannot reflect dynamic factors of the resource performance. However, if the workloads of grid resources are not very heavy, it can still give a good approximation and greatly simplify the system implementation. The resource information will also be simpler, and referring to the processor type is sufficient. In some situations, the processors of a grid resource are homogenous. In this case, there is no need to give a list of processors. Just giving the number of processors and corresponding processor type is enough. The processors of a grid resource can be expressed as follows:

$$P = \{P_i | i = 1,2,......,n\}$$
$$ty = \{ty_i | i = 1,2,......,n\}.$$

Let m be the number of applications that are running, or being queued to be executed on a grid resource. Existing applications on a resource will impact the resource performance. If there are a lot of applications queued for a resource, the resource may have little chance to meet requirements from future requests. The application information includes a list of applications that are running or queued on a resource. Each application A_j has two attributes: scheduled start time ts_j, and end time te_j. The applications of a grid resource can be expressed as follows:

$$A = \{A_j \mid j = 1,2,......,m\}$$
$$ts = \{ts_j \mid j = 1,2,......,m\}$$
$$te = \{te_j \mid j = 1,2,......,m\}.$$

The application-resource mapping gives a map of how processors of a resource are allocated to applications. Let MA_j be the set of processors that are allocated to application A_j:

$$MA = \left\{ MA_j \,|\, j = 1,2,......,m \right\}$$
$$MA_j = \left\{ P_{i_l} \,|\, l = 1,2,......,k_j \right\},$$

where k_j is the number of processors that are allocated to application A_j. Let M be a 2D array, which describes the mapping relationships between resources and applications using Boolean values.

$$M = \left\{ M_{ij} \,|\, i = 1,2,......,n;\, j = 1,2,......,m \right\}$$
$$M_{ij} = \begin{cases} 1 & if \quad P_i \in MA_j \\ 0 & if \quad P_i \notin MA_j \end{cases}$$

The contents of service information are described above. The ACT manager is also responsible for maintenance of different kinds of ACTs according to different strategies described in Table 5.1. The service advertisement in ARMS is performed in the same way as described in the A4 methodology.

6.3.2 PACE Evaluation Engine

As mentioned in Section 5.4, a request is composed with request information, requirements, and additional options. In ARMS, a request for service discovery is to find an available grid resource for an application.

The request information is basically the PACE application model *am*, which includes all of the performance related information of an application A_r. The application model will be one of the inputs to the PACE evaluation engine in an agent.

The requirements in ARMS are specified in a cost model, which can include many metrics, for example, the deadline for the execution of an application to be finished, t_{req}. The cost model is one of the inputs to the matchmaker in an agent.

The PACE evaluation engine has two inputs, the application model from the request, *am*, and the resource information from the ACT manager, *ty*. Using this information, the PACE evaluation engine can produce performance prediction information such as application execution time, *exet*, for the application to be executed on the given resource.

$$exet = eval(ty, am)$$

Instead of running the application on all of processors of a grid resource *P*, an application can choose to be executed on any subset of processors $\overline{P}$ (note that $\overline{P}$ cannot be an empty set Φ), which can also be evaluated and expressed as follows:

$$\forall \overline{P} \subseteq P, \overline{P} \neq \Phi, \overline{ty} \subseteq ty, \overline{ty} \neq \Phi, \overline{exet} = eval(\overline{ty}, am).$$

The output of the PACE evaluation engine, *exet*, is one of the inputs to the scheduler of the agent. Another input to the scheduler is the application information from an ACT item.

6.3.3 Scheduler

An ACT item acts as a vision of a grid resource that is remote to the agent. However, an agent can still schedule the required application execution based on this information of a resource. The function of the scheduler is to find the earliest time for an application to be finished on the resource described by an ACT item, t_{sched}.

$$t_{sched} = \min_{\forall \overline{P} \subseteq P, \overline{P} \neq \Phi} \left(\overline{te_r} \right)$$

The application has the possibility of being allocated to any selection of processors of a grid resource. The scheduler should consider all of these possibilities and choose the earliest end time of the application execution. In any of these situations, the end time is equal to the earliest possible start time plus the execution time, which is described as follows:

$$\overline{te_r} = \overline{ts_r} + \overline{exet}\,.$$

The earliest possible start time for application A_r on a selection of processors is the latest free time of all of selected processors if there are still applications running on the selected processors. If there is no application currently running on the selected processors, application A_r can be executed on these processors immediately. These can be expressed as follows:

$$\overline{ts_r} = \max\left(t, \max_{\forall i, P_i \in \overline{P}}\left(td_i\right)\right),$$

where td_i is the latest free time of processor P_i. This equals to the maximum end time of applications that are allocated to process P_i:

$$td_i = \max_{\forall j, M_{ij}=1}\left(te_j\right).$$

In summary, t_{sched} can be calculated as follows:

$$t_{sched} = \min_{\forall \overline{P} \subseteq P, \overline{P} \neq \Phi}\left(\max\left(t, \max_{\forall i, P_i \in \overline{P}}\left(\max_{\forall j, M_{ij}=1}\left(te_j\right)\right)\right) + \overline{exet}\right).$$

It is not necessarily the case that scheduling all processors to an application will achieve higher performance. On the one hand, the start time of application execution may be earlier if only a number of processors are selected; on the other hand, with some applications, execution time may become longer if too many processors are allocated.

The scheduling algorithm described above is used in the initial implementation of ARMS. The complexity of the algorithm is determined by the number of possible processor selections, which can be calculated as:

$$C_n^1 + C_n^2 + \ldots\ldots + C_n^n = 2^n - 1$$

It is clear that if the number of processors of a grid resource increases, the complexity of the local resource scheduling algorithm will increase exponentially. Though a local resource in a grid environment can only have limited number of processors, this algorithm cannot scale well when the number of processors increases. Another factor is that the scheduling policy of this algorithm is to meet requirements from the user, instead of maximising the resource utilisation. There is no rescheduling process for previously scheduled applications. New algorithms need to be developed in a practical implementation of ARMS; this will be discussed in Chapter 8.

We can also see the importance of the efficiency of the PACE evaluation engine. During each scheduling process, the evaluation function can be called 2^n-1 times. Even in the situation where all the processors of a grid resource are of the same type, the evaluation function still needs to be called *n* times. PACE evaluation can be performed very quickly to produce prediction results on the fly, which is the key feature for PACE to be used in ARMS to provide QoS support for service discovery.

6.3.4 Matchmaker

The matchmaker in an agent is responsible for comparing the scheduling results with the cost model attached to the request. The comparison results lead to different decisions on agent behaviours according to service discovery strategies described in Section 5.4.1.

In terms of application execution time, if $t_{req} \geq t_{sched}$, the corresponding resource can meet the user requirement. If the corresponding ACT item is in the T_ACT, a local resource is available for application execution. The application execution command will be sent to the local management in the agent. Otherwise, the agent ID of the corresponding ACT item is returned, and the agent will dispatch the request to that agent via the agent ID.

If $t_{req} < t_{sched}$, the corresponding resource cannot meet the requirement from the user. The agent continues to look up other items in ACTs until the available service information is found. The agent can look up different ACTs in turn. If there is no available service information in ACTs any more, the agent may submit or dispatch the request to its upper or lower agents for further discovery according to its own strategy configurations for service discovery.

There may be many other metrics in the cost model from the user. The corresponding evaluation mechanisms should also be provided in each agent. Their implementation will be the same as the application execution time described in this section. These will not be discussed in detail here.

6.4 ARMS Implementation

ARMS has been developed to demonstrate how the A4 methodology is coupled with PACE functions to achieve grid resource management. Each ARMS agent is composed with an agent kernel and some agent information browsers. A case study is given and some experimental results are also included to show how ARMS schedules applications onto available resources.

6.4.1 Agent Kernel

The kernel of each agent is developed in C/C++ and fulfils all of the main functions described in the last sections. The agent kernel makes extensive use of the file system, and a collection of various database files representing its complete state at any particular instant in time.

The most important file in an agent is the log file. After an application execution request is received in an agent, it undergoes a series of state changes, with each state representing a particular stage in its lifetime. The various states for a request to be processed in an agent include: queuing, discovering, waiting, running, submitted, etc.

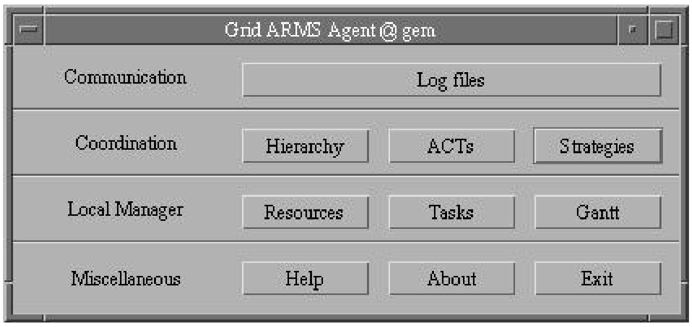
The agent hierarchy database file is used to record the contact IDs of the upper and lower agents. There are also various database files used as agent ACTs. A separate thread in an agent exists for service advertisement and ACT maintenance according to the strategy configurations, which are also stored in a separate database file.

At a local management level, resource and application information are represented in different database files. As mentioned before, the system focuses on agent coordination and meta-level service advertisement and discovery. Though there is related information existing in the local management layer of each agent, applications fake executing on corresponding resources, which does not impact on the system performance being investigated and simplifies the system implementation.

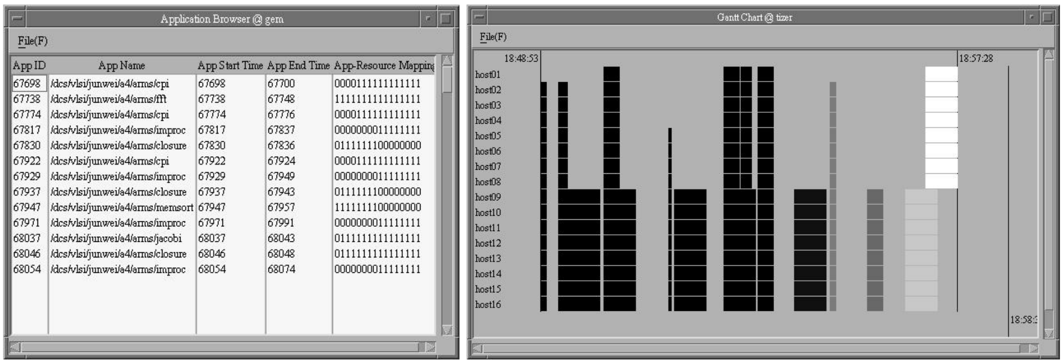
6.4.2 Agent Browser

One of the main goals of the initial implementation of ARMS is to make the state of the system visible and enable the performance of the system to be investigated. Agent browsers are developed using the X windows library and can be used to show all contents of the database files within an agent described in the last section. These are all illustrated in Figure 6.5.

Each agent has an operational platform, which includes a menu for activating various agent browsers, shown in Figure 6.5(a). Figure 6.5(b) shows an example of agent browsers, an application browser, which gives details of applications that are running or queuing on the local resource. A Gantt chart is also designed to give a graphical interface to visualise the make spans of all the applications shown in the application browser, which is illustrated in Figure 6.5(c).



(a) Operational platform



App ID	App Name	App Start Time	App End Time	App-Resource Mapping
67698	/dcs/vlsi/junwei/a4/arms/cpi	67698	67700	0000111111111111
67738	/dcs/vlsi/junwei/a4/arms/fft	67738	67748	1111111111111111
67774	/dcs/vlsi/junwei/a4/arms/cpi	67774	67776	0000111111111111
67817	/dcs/vlsi/junwei/a4/arms/improc	67817	67837	0000000011111111
67830	/dcs/vlsi/junwei/a4/arms/closure	67830	67836	0111111100000000
67922	/dcs/vlsi/junwei/a4/arms/cpi	67922	67924	0000111111111111
67929	/dcs/vlsi/junwei/a4/arms/improc	67929	67949	0000000011111111
67937	/dcs/vlsi/junwei/a4/arms/closure	67937	67943	0111111100000000
67947	/dcs/vlsi/junwei/a4/arms/memsort	67947	67957	1111111100000000
67971	/dcs/vlsi/junwei/a4/arms/improc	67971	67991	0000000011111111
68037	/dcs/vlsi/junwei/a4/arms/jacobi	68037	68043	0111111111111111
68046	/dcs/vlsi/junwei/a4/arms/closure	68046	68048	0111111111111111
68054	/dcs/vlsi/junwei/a4/arms/improc	68054	68074	0000000011111111

(b) Application browser

(c) Gantt chart

Figure 6.5 ARMS Agent Browsers

Agent browsers are updated in real time when the system is running. The user can also change the strategies to configure the agent with a different behaviour for service advertisement and discovery from the strategy browser. The agent behaviours can also be configured using the PMA semi-automatically, which will be discussed in Chapter 7.

6.5 A Case Study

Experiments have been designed using the initial implementation of ARMS. There are two main parts in the design of the experiments. ARMS itself includes agents, resources, and agent behaviour strategies used in the experiment. The automatic users of the system are also designed to send application execution requests to ARMS with different frequencies, which add different workloads onto the system.

6.5.1 System Design

There are 8 agents in the experimental system. The agent hierarchy is shown in Figure 6.6. The agent at the head of the hierarchy is *gem*, which has three lower agents: *sprite*, *origin*, and *tizer*. The agent *origin* has no lower agents, while *sprite* and *tizer* have two lower agents each.

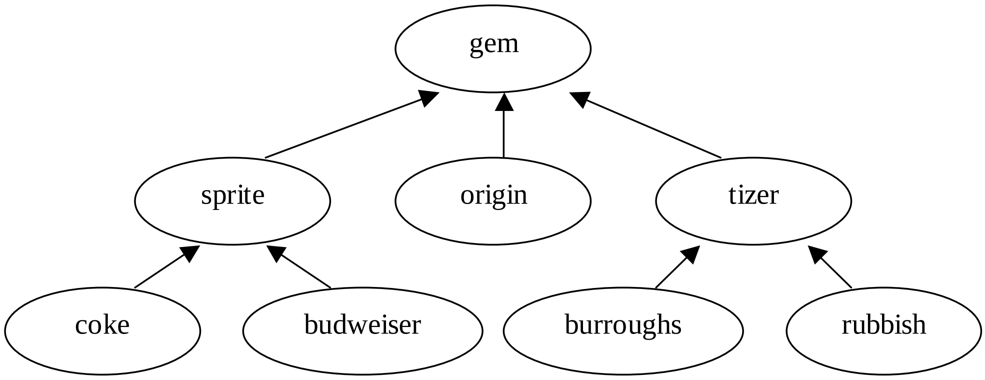


Figure 6.6 ARMS Case Study: Agent Hierarchy

Each agent is a representative of a local grid resource. The information of the resources is shown in Table 6.1. Each resource is composed with 16 processors (for SGI) or hosts (for Sun), and each host has the same resource type. The SGI multi-processor is the most powerful, followed by the Sun Ultra 10, 5, 1, and SparcStation in turn.

Agent	Resource Type	#Processors/Hosts
gem	SGI Origin 2000	16
origin	SGI Origin 2000	16
sprite	Sun Ultra 10	16
tizer	Sun Ultra 10	16
coke	Sun Ultra 1	16
budweiser	Sun Ultra 5	16
burroughs	Sun SPARCstation 2	16
rubbish	Sun SPARCstation 2	16

Table 6.1 ARMS Case Study: Resources

In the experimental system, the T_ACT, L_ACT and G_ACT are used in each agent. T_ACTs are maintained by the event-driven data-push service advertisement. L_ACTs are updated once every 10 seconds using periodical data-pull. G_ACTs are updated once every 30 seconds using periodical data-pull. All of the agents use the same strategies except that *gem* is the head of the agent hierarchy and does not maintain a G_ACT. The choice of different strategies impacts on the service discovery performance of the overall system, which will be discussed in detail in Chapter 7.

The agents and resources have been defined and configured above, while another important design aspect of the experiment is the requests. To add workloads automatically to ARMS, we design virtual users that send application execution requests to the agent system.

6.5.2 Automatic Users

The applications that are used in the experiment are some typical scientific computing programs, including sweep3d, fft, improc, closure, jacobi, memsort, and cpi. Each application has been modelled and evaluated using the PACE toolkit. The performance evaluation results against the SGI Origin2000 can be found in Figure 6.7. The run time spent on other platforms is much more than that on the SGI Origin2000, but the trend of the curve is almost the same, which is not shown in details.

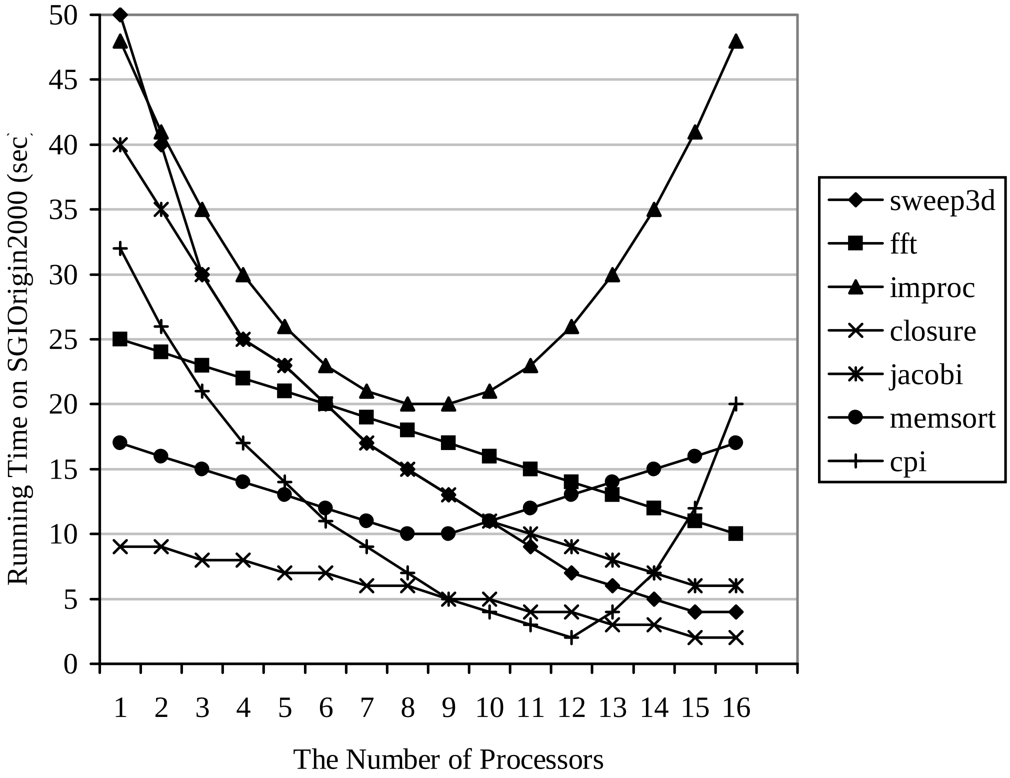


Figure 6.7 ARMS Case Study: Applications

Each request chooses one of the 7 applications randomly and is sent to one of 8 agents randomly. The required execution time for the application is also chosen randomly from a given domain, which is described in Table 6.2.

Application	Minimum Requirement (s)	Maximum Requirement (s)
sweep3d	4	200
fft	10	100
improc	20	192
closure	2	36
jacobi	6	160
memsort	10	68
cpi	2	128

Table 6.2 ARMS Case Study: Requirements

The automatic users can be configured to send requests with different frequencies. As shown in Table 6.3, four experiments are designed with different workloads added to ARMS. The interval of requests sent in each experiment is chosen randomly from a given domain, which results in a different average frequency. For example, experiment No. 2 lasts about 7 minutes. During this period, 149 requests are sent to ARMS. There is one request sent every 3 seconds on average. The experimental results will be discussed in the sections below.

Experiment No.	1	2	3	4
Minimum Request Interval (s)	1	1	1	1
Maximum Request Interval (s)	7	5	3	1
Average Frequency (s/application)	4	3	2	1
Experiment Last Time (min)	7	7	7	5
Total Application Number	109	149	215	293

Table 6.3 ARMS Case Study: Workloads

6.5.3 Experiment Results I

In this section, the detailed results of experiment No. 2 are given. In this experiment, there are a total of 149 applications scheduled to be executed on 8 resources. The detailed results are listed in Appendix B, which can be illustrated using both a user's (global) view and agent (local) views.

A request is submitted by the user to ARMS with a requirement of execution time. Agents in ARMS cooperate with each other to find an available resource that can meet the user requirement. The service discovery process can be completed in 0, 1, or 2 steps. For example, in a 2-step service discovery, three agents are involved. The first agent receives the request from the user, the corresponding resource is found at the final agent, and the second acts as a go-between during the process. The application execution results are returned, including the time spent on discovery, waiting, execution, etc.

The experimental results shown in Appendix B also give a list of application execution data in the local management layer of each agent. An agent schedules the accepted application executions onto the local resource. The corresponding information includes the start time, the end time, and the mapping between the application and the processors/hosts. These can be illustrated clearly using Gantt charts.

The detailed results of this single experiment show how ARMS uses agent-based service advertisement and discovery to achieve grid resource management. However, the capability and performance for agents to schedule applications onto grid resources can be only illustrated by the comparison of the statistical data from several experiments. These are discussed below.

6.5.4 Experiment Results II

In this section, some statistical data on the results of the four experiments is given. Note that the detailed results for the other three experiments are not given and only statistical data are included in the tables below.

The distributions of the application execution against agents for all the experiments are summarised in Table 6.4. For example, in the experiment No. 2, 27 requests of the application execution are scheduled onto the resource of the agent *gem* (this is conformed with the detail results shown in Appendix B.2), which are 19 percent of the total 149 requests. 5 requests (3 percent of the total requests) are not scheduled onto any resource and end unsuccessfully (this is summarised from the data shown in Appendix B.1).

Agent	Experiment Number							
	1		2		3		4	
	No.	%	No.	%	No.	%	No.	%
gem	13	12	27	19	45	21	45	15
origin	13	12	15	10	27	13	42	14
sprite	15	14	20	13	27	13	38	13
tizer	14	13	27	19	31	14	39	13
coke	10	9	15	10	20	9	28	10
budweiser	13	12	17	11	23	11	31	11
burroughs	14	13	12	8	16	7	26	9
rubbish	14	13	11	7	17	8	24	8
failed	3	2	5	3	9	4	20	7
Total	109	100	149	100	215	100	293	100

Table 6.4 ARMS Experiment Results: Application Execution

The distributions of the application execution against service discovery for all the experiments are summarised in Table 6.5. For example, in the experiment No. 2, the resources for 114 requests of the application execution are discovered immediately at the agent they are submitted to, which are 77 percent of the total 149 requests. This can also be summarised from the data shown in Appendix B.1.

#Step	Experiment Number							
	1		2		3		4	
	No.	%	No.	%	No.	%	No.	%
0-step	106	97	114	77	143	66	199	68
1-step	3	3	24	16	38	18	29	10
2-step	0	0	11	7	31	15	53	18
3-step	0	0	0	0	3	1	12	4
Total	109	100	149	100	215	100	293	100

Table 6.5 ARMS Experiment Results: Service Discovery

The statistical results shown in Tables 6.4 and 6.5 are also illustrated in Figures 6.8 and 6.9 respectively. The curves in the figures show trends of the application distributions when the system workload increases. These are also discussed in detail below.

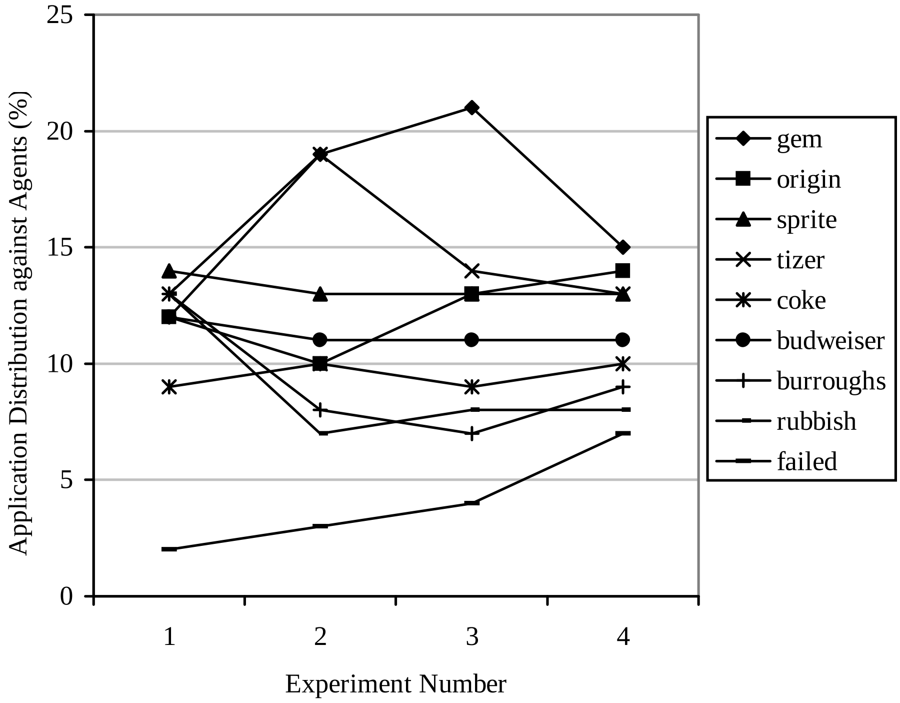


Figure 6.8 ARMS Experiment Results: Application Execution

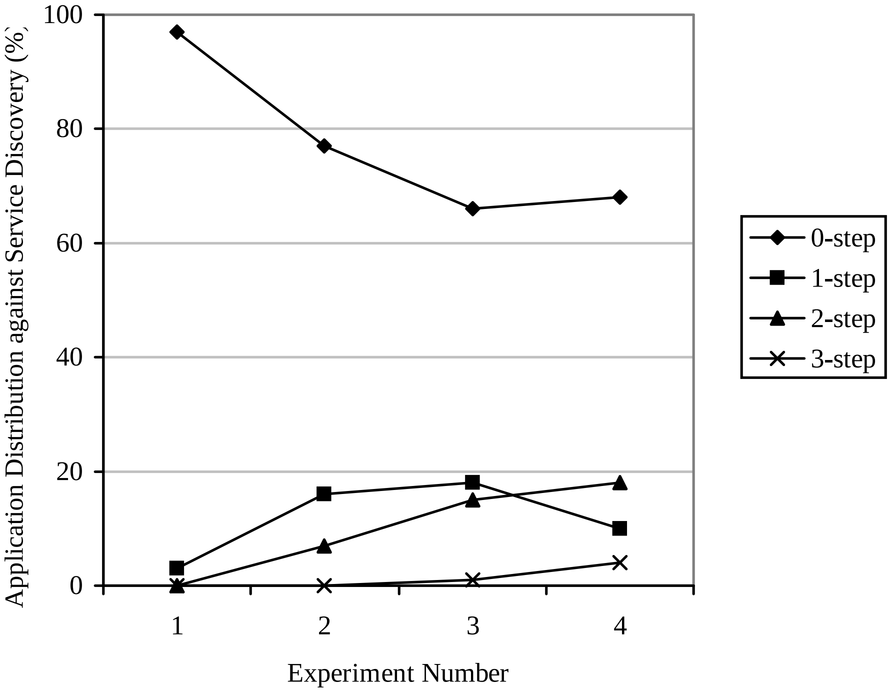


Figure 6.9 ARMS Experiment Results: Service Discovery

1. There is one request sent every 4 seconds on average in the experiment No. 1. Application execution requests are sent to the agents randomly, so each agent should receive the same number of requests from users. In this experiment, the system workload is rather light in relative to the capabilities of the resources (even to the resources of agent *burroughs* and *rubbish*, which are not so powerful). The 97% 0-step discoveries show that almost all the requests are met immediately at the first agent they arrive. Almost no service discovery processes occur between agents. This results in an average application distribution on the agents and the number of the requests that end unsuccessfully is very small.

2. The local resources of agent *burroughs* and *rubbish* are clusters of Sun SPARCstation 2, which is not as powerful as the other platforms existing in other agents. In the experiment No. 2, when the system workload becomes heavier, many requests that they cannot meet are submitted to their upper agent, *tizer*, which leads to a very heavy workload on *tizer* (19% of application executions). The resources of agent *coke* and *budweiser* are a bit more powerful. However, they still cannot meet all of the requests from users. Some of the requests are submitted to their upper agent, *sprite*, which leads to a heavy workload on *sprite*, though not so heavy as *tizer*. These result in the dramatic increase of the percent of 1-step service discovery processes. The agent *gem* is the head of agent hierarchy and has the most powerful platform, a multi-processor SGI Origin2000. There are some application execution requests that have very critical requirements. These requests can only be met using the SGI Origin2000, so are submitted from *tizer* or *sprite* to *gem*. This leads to a rather heavy application execution workload on *gem* and also an increase of the processes for the 2-step service discovery. However, as shown in the Gantt chart of *gem* in Appendix B.2, *gem* is so powerful that it is still not fully utilised. The resource of another agent, *origin*, is as powerful as that of *gem*, and can meet all of the requests it receives from users. However, *origin* is a little far from the other agents. This results in the fact that *origin* is far from utilised, which is also illustrated in the Gantt chart of *origin* in Appendix B.3.

3. The system workload increases further. The dramatic decrease of the percent of application executions on *tizer* indicates that the local resource of *tizer* is fully utilised in this situation. Many requests submitted from *burroughs* and *rubbish* have to be passed to *gem*, which leads to a dramatic increase of the number of 2-step discovery processes. The number of 1-step discovery processes increases too and a few 3-step discovery processes appear. More application executions are scheduled onto the agent *origin*. All of these indicate that service discovery among the agents becomes active when the system workload increases.

4. The system workload becomes very heavy in this situation. The decrease of the percent of application executions on *gem* indicates that the local resource of *gem* also reaches its capability limitation, which results in a dramatic increase of the number of the unsuccessful requests. The number of 1-step discovery processes decreases, while 2-step and 3-step service discovery processes occur more often. All of these indicate that the whole system is fully utilised, so more complex service discovery processes occur in order to find the available resources for the requests. However, in this situation, the application executions show a very reasonable distribution against agents. The order of the workload on the agents is the same as that of the computing capabilities of their resources. The agent *gem* and *origin*, which represent the most powerful resources in the example system, have more applications executed, followed by *sprite, tizer*, *budweiser* and *coke*. And only a small number of requests are met at the agent *burroughs* and *rubbish*.

These experimental results show that the performance prediction driven agent-based service advertisement and discovery is effective for the applications to be scheduled at the meta level to utilise the grid resources. As we have mentioned in Section 2.4, scalability and adaptability are two key challenges that the implementation of grid resource management must address.

As shown in the experimental results, agents are organised into a hierarchy and only process service advertisement and discovery with nearby agents. Once the computing power in a scope cannot meet the requirements received, the additional requests will be gradually dispatched to a larger scope, where the workload is not so heavy compared with the computing capabilities. Note that the service discovery is not processed in one step, but step by step, and may bypass many intermediate agents. This key feature makes it possible for the system to scale well when the grid environment becomes very large.

The PACE performance evaluation functions are used in the ARMS implementation both locally and remotely. In order for an agent to make decisions, the PACE evaluation engine will be called many times. The rapid evaluation time of PACE without sacrificing accuracy is a very important feature for the ARMS implementation.

Another important factor for ARMS to achieve high performance is the capability for the agents to adjust their behaviours for service advertisement and discovery to adapt to the highly dynamic grid environment. Though some of the strategies have been introduced in the A4 methodology, and a PMA is also included in the ARMS architecture, meta-level performance optimisation of ARMS using PMA is not discussed in detail. In the next chapter, the implementation of PMA is described to provide ARMS with high adaptability.

Chapter 7

PMA: PERFORMANCE MONITOR AND ADVISOR FOR ARMS

Performance issues arise from the dynamic nature of grid resources [Cao2001]. As we have mentioned in the A4 methodology, most practical systems must make a balance between service advertisement and discovery. The PMA is a special agent, which is capable of performance modelling and simulation about the agent system and acts as a performance monitor and advisor in ARMS. In this chapter, the structure for the PMA implementation is described along with details on performance optimisation strategies and steering policies. A case study is also used to show how different strategies and policies are used to improve the performance of the ARMS agent system.

7.1 PMA Structure

The PMA was illustrated in Figures 6.1 and 6.2 previously. Unlike facilitators or brokers in classical agent-based systems it is not central to the rest of the agents. It neither controls the agent hierarchy nor serves as a communication centre in the physical and symbolic sense. Instead, the PMA observes the communication traffic of the agent system and tries to draw corresponding conclusions regarding the agents' behaviour with the intention of improving the performance of ARMS.

If the PMA ceases to function, the agent system has no difficulty in surviving and it continues with its ordinary life. The efficiency improvement consideration would not be provided in ARMS unless some modelling and simulation mechanism is built into the PMA. By introducing the PMA, we have tried to avoid making ARMS unscalable by relying on a single agent, which otherwise becomes a system bottleneck.

In this section, we will introduce the structure of the PMA and its relation with other agents in ARMS, which is shown in Figure 7.1. The kernel of the A4 simulator is used in the PMA, including the model composer and simulation engine. However, the PMA has a different way of input and output.

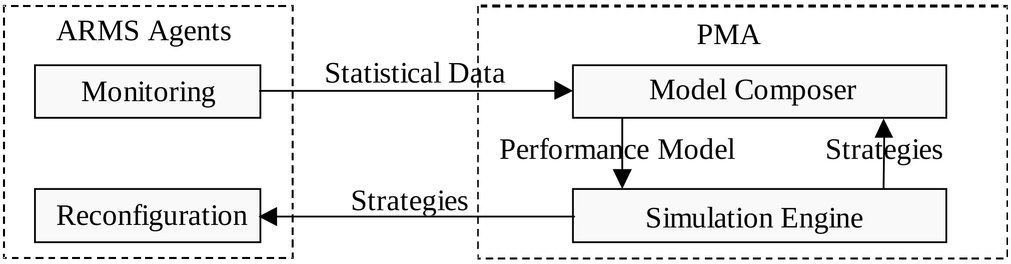


Figure 7.1 PMA vs. ARMS

Statistical data is monitored from each of the ARMS agents and input to the PMA for performance modelling. As introduced in Section 5.6.1, the statistical data that are input into the model composer mainly concern the requests and services in the system. These include:

- Relative request performance value. In ARMS, this value is the required application execution time.
- Request sending frequency. An agent may receive the same request from a user very frequently. The PMA sensors in the ARMS agents can analyse the request information in the log file and calculate the average time a request is received.
- Relative service performance value. In ARMS, the resource performance is evaluated using the PACE toolkit and scheduling algorithms, which makes the modelling and simulation very difficult. Some estimation on

average application waiting and execution time can be used as a relative service performance value.

- Service performance changing frequency. The grid resources are dynamic and their performance varies over time. The PMA sensor in the ARMS agents can monitor the updating frequency of T_ACT and estimate an average performance changing frequency.

The statistical data and other relative information are composed into a performance model. The performance model is put through the simulation engine in the PMA. The optimisation strategies used in ARMS to improve service discovery will be discussed in Section 7.2. New optimisation strategies can be chosen to improve the performance metrics according to some steering policies, which will be discussed in Section 7.3. The simulation can be performed many times until a better solution is selected. The selected optimisation strategies are returned and used to reconfigure the agents in ARMS.

7.2 Performance Optimisation Strategies

When the A4 methodology and the ARMS implementation were introduced earlier, some strategies for ACT maintenance were discussed. However, the impact of the choice of these strategies on the overall system performance is not discussed in detail. There are also further performance optimisation strategies that can be considered, which will be discussed in detail below.

7.2.1 Use of ACTs

T_ACT is always used in each agent and cannot be used for service discovery performance optimisation, because the connections made between the local resource and the T_ACT in the agent take place within an agent and have no effect on communications between agents.

Caching previous service discovery results is a good strategy for performance optimisation that assumes a request may be required more than once. Many current network applications use caches to optimise performance. Using cached service information may result in direct service discovery in one step. However, if the service information changes frequently compared to the request frequency, using the cache may decrease the service discovery speed. So the efficiency of using cache depends on the characteristics of the actual system.

Adding some local knowledge to an agent is also a performance optimisation that assumes that services are often required by local agents. If an agent has the service information of its lower agents, it need not traverse all of them for service discovery and dispatch the request to the available lower agent directly. However, additional data maintenance workload is needed for the L_ACT.

Adding some global knowledge to an agent is also a performance optimisation. A request may need less connections to find the available service as the higher-level agents need not take part in the discovery process. The system load can also be reduced. Additional data maintenance workload is also needed for the G_ACT.

The efficiency of using L_ACT and G_ACT also depends on the characteristics of the actual system. Balance must be made between service advertisement and discovery when L_ACT and G_ACT are used in agents. How to steer the performance optimisation process will be discussed in Section 7.3 and illustrated using a case study in Section 7.4.

7.2.2 Limited Service Lifetime

Another performance optimisation strategy is to add a service lifetime limitation to the attributes of the service information. This lifetime should be pre-estimated before the service is advertised. The agent can check the ACTs frequently and delete out-of-date service information. This can avoid unnecessary routing processes and increase the speed of service discovery. There is also no additional data maintenance workload. However, the lifetime of some services in the system may be unpredictable.

7.2.3 Limited Scope

The scope in which a service can be advertised and discovered can also be pre-defined by attributes to the service information. The service need only be advertised within a certain scope of the system, which can reduce the advertisement and data maintenance workload. The search for a service can also be limited to a certain scope, avoiding unnecessary discovery processes. However, a prior knowledge about the service and its requests are needed to achieve optimisation. Mismatches between the scope limitation of a service and of a request may result in the low success rate of the service discovery.

7.2.4 Agent Mobility and Service Distribution

A good match between the requests and services in the system may lead to higher performance service discovery. For example, in the grid environment, if a scope with many requests has also many high performance computing resources, these requests need not be routed far away to find an available resource, which decreases the service discovery workload. However, request distribution is up to the users and cannot be changed by the system. So agent mobility and service re-distribution can be used to give a better match with the requests.

The case study in Section 5.7 provides a good illustration that agent movement and service re-distribution can lead to a higher performance. When the service is moved to a coordinator of a larger sub-hierarchy, more requests become local instead of remote, which reduces the discovery workload of the system.

It is clear whether the strategies described above can be used to improve performance is determined by the characteristics of the system. The performance of the system may vary when the grid resources change. So the process for the PMA to monitor and reconfigure the ARMS agent exists during the lifetime of the system. When the system states change, the PMA is responsible for changing the performance optimisation strategies and configuring the relative agent behaviours to adapt to the new situation. Some performance steering policies can be used to guide the changing of strategies and configure the agent behaviours to achieve higher service discovery performance gradually. These will be discussed in the following sections.

7.3 Performance Steering Policies

There are four metrics that are used to characterise the performance of the system, which were given in Section 5.5. The processes for the PMA to steer the performance of the ARMS agents are driven by improving these metrics. Different systems have different critical aspects and have different criteria of high performance. In this section, we focus on load balance between the service advertisement and discovery, which is commonly needed in most of the systems.

Discovery speed (*v*) and system efficiency (*e*) are two metrics that are used for load balancing between the service advertisement and discovery. The system is in very low performance mode when both discovery speed and system efficiency are very low. In this situation, agents can be steered and configured with more service advertisement. Reasonable service advertisement can lead to less workload on service discovery and improve both discovery speed and system efficiency simultaneously. However, too much service advertisement may decrease system efficiency though increase discovery speed. Let’s consider each kind of ACT maintenance approach.

Each agent maintains a T_ACT in its coordination layer, which includes the service information of the local resource. Periodically updating T_ACT may save update workload but cause delay on updating and unnecessary trouble for service discovery. Maintenance of T_ACT does not add workload on agent communication, so event-driven updating can be used to keep T_ACT in line with resource changes in real time. Because event-driven data-pull updating of T_ACT may increase service discovery time, it is better to use an event-driven data-push approach to keep the T_ACT updated in real time. However, if the resource changes very frequently and the number of requests is very small, an event-driven data-pull approach can also be used.

In most situations, cached information can improve the system performance to some extent. Especially when system performance is very low and there is no cached information maintained in each agent, adding C_ACT in each agent of the system could result in obvious performance improvement. In general, C_ACT is maintained using both event-driven data-pull and data-push approaches, which is the same as other kinds of use of cache.

Keeping some service information of lower agents can always improve service discovery performance if the system is not extremely dynamic. However, in general only one of the four approaches can be chosen for L_ACT maintenance. Two or more approaches applied at the same time may cause redundancy of service advertisement. The service discovery may not benefit from the redundant service information enough so that the system efficiency may decrease.

Use of G_ACT has the same policies as L_ACT. Note that data-push updating of G_ACTs should be applied to the system carefully. Because the updating takes place in all of the lower agents of an agent, the service advertisement workload could increase greatly. However, the lower agents may not make good use of this updated service information for service discovery, which leads to a low system efficiency.

Another advantage of using G_ACT, is to avoid adding too much service discovery workload to the coordinators or the broker in the agent hierarchy and improve load balance of the agents. The success rate of the system can also be improved using available limit service lifetime and scope configurations. This is not discussed in detail here.

In fact, it is difficult to define obvious and efficient policies to guide the performance optimisation processes used in PMA. There are too many factors that have an impact on system performance and whether a strategy can be chosen to improve performance depends heavily on the real situation of the system. The system can be steered at a global level, which means that all of the agents are configured with the same strategies. However, each agent can also be configured with a different strategy. In this section, we only discuss the problem of performance steering initially. Further research is needed to give a deeper analysis of the performance optimisation issues.

7.4 A Case Study

In this section, an example model is given and experiment results are included to show how to steer the performance optimisation process using the PMA. Note that the simulation results included in this section are actually produced using the A4 simulator.

7.4.1 Example Model

The attributes of an example model are shown in several tables. This is composed of about 250 agents, each representing a high performance computing resource that may provide a computing capability with a different performance. These agents are organised in a hierarchy, which has three layers. The identity of the root agent is *gem*. There are 50 agents registered to *gem*, four of which each also have 50 lower agents. The hierarchy is illustrated in Table 7.1.

Agents	Upper Agent
gem	-
sprite~0……sprite~49	*gem*
tup~0……tup~49	*sprite~9*
cola~0……cola~49	*sprite~19*
tango~0……tango~49	*sprite~29*
pepsi~0……pepsi~49	*sprite~39*

Table 7.1 Example Model: Agents

To simplify the modelling processes, we define the services and requests in the agents at the system level, which is shown in Table 7.2 and 7.3 respectively. The name of the services and requests are all HPC, but with different relative performance values. The frequency value of the service, 5, for example, means the service performance will change between 0 and the performance value once every 5 steps during the simulation. The frequency value of the request, 5, for example, means a request will be sent once every 5 steps during the simulation. A step can be designed as an arbitrary number of seconds. In ARMS, these values must be monitored by the PMA while the system is operational. The performance optimisation strategies of the lifetime and scope limitations are not used in the model. The distribution value is used to define how many agents will be configured with the corresponding service or request.

Name	Relative Performance	Freq	Lifetime	Scope	Dist (%)
HPC	1000	5	Unlimited	Top	20
HPC	600	10	Unlimited	Top	40
HPC	200	20	Unlimited	Top	60

Table 7.2 Example Model: Services

Name	Relative Performance	Freq.	Scope	Dist. (%)
HPC	100	5	Top	80
HPC	300	10	Top	60
HPC	500	20	Top	40
HPC	800	40	Top	20
HPC	1000	60	Top	10

Table 7.3 Example Model: Requests

Finally, the model must define how each agent uses the ACTs to optimise the performance. In this case study six experiments have been considered, each of which has the same configurations as described in Table 7.1 – 7.3, but has different optimisation strategies as described in Table 7.4.

Performance Optimisation Strategies	Experiment Number					
	1	2	3	4	5	6
T_ACT: event-driven data-push	✓	✓	✓	✓	✓	✓
C_ACT: event-driven data-push and data-pull		✓	✓	✓	✓	✓
L_ACT: event-driven data-push			✓	✓	✓	✓
G_ACT: periodic data-pull every 10 steps				✓	✓	✓
L_ACT: periodic data-pull every 10 steps					✓	✓
G_ACT: event-driven data-push						✓

Table 7.4 Example Model: Strategies

To simplify the experiments, we only define the strategies at the system level, which means all of the agents in the model must use the same performance optimisation strategies. A mixture of optimisation strategies is possible but is not considered in these experiments. In the simulation results included in Section 7.4.2, a comparison of the different strategies is given by considering their impact on the system performance.

7.4.2 Simulation Results

The simulation results for all of the experiments are summarised in Table 7.5. Note that all values are accumulative results after 200 simulation steps. Each of the six situations are described in detail below.

Metrics	Experiment Number					
	1	2	3	4	5	6
r	12296	12355	12576	12560	12645	11715
a	0	0	5604	8051	10172	285148
d	65595	51113	7435	6901	6910	7056
v	0.18	0.24	1.69	1.82	1.82	1.84
e	0.18	0.24	0.96	0.84	0.74	0.04

Table 7.5 Simulation Results

1. Only T_ACTs are used in each agent. Each time the request arrives, a lot of connections must be made and traversed in order to find the satisfied service. In this situation, the discovery speed and system efficiency are both rather low.

2. The cache is used in each agent, which needs no extra data maintenance and improves the discovery speed and system efficiency a little. This is because the dynamics of the services reduce the effects of the cached information and so becomes unreliable.

3. L_ACT is added in each agent. Each time the service performance changes, the corresponding agent will advertise the change upward in the hierarchy. This adds additional data maintenance workload to the system,

which decreases the discovery workload extremely. So the discovery speed and the system efficiency are all improved.

4. G_ACT is also added. Each agent will get global service information from its upper agent once every 10 simulation steps, which will add additional data maintenance workload. From the simulation results, we can see this improves the discovery speed further. But the system efficiency decreases a little because of the additional data maintenance.

5. Another maintenance of the L_ACT is added. Each agent asks for service information from its lower agents once every 10 steps. This doesn't improve the discovery speed any more and only adds more data maintenance workload, which decreases the system efficiency further.

6. Another maintenance of the G_ACT is added. This improves the discovery speed only a little, but adds further data maintenance workload, which decreases the system efficiency extremely.

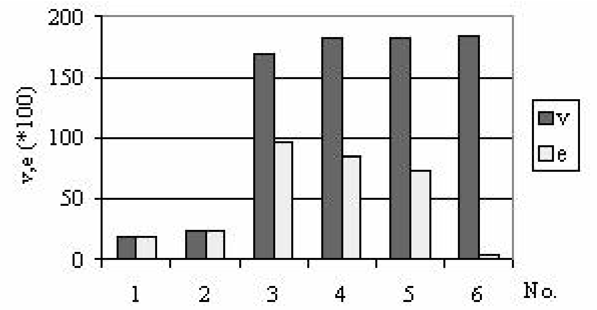


Figure 7.2 Choice of Optimisation Strategies

The impact of the choice of the optimisation strategies on the discovery speed and the system efficiency is shown clearly in Figure 7.2. It can be seen that the fourth experiment has a good balance between the discovery speed and the system efficiency for this example model. It has a higher discovery speed in comparison to the third, with only slight lower system efficiency.

Changing the G_ACT update frequency will also change the performance of the model. Figure 7.3 shows the relation between the G_ACT update frequency and the system performance. In these experiments, the strategies that are used are all the same as described in the fourth experiment of Table 7.4. The only difference is the G_ACTs in the agents are updated with different frequencies, which may lead to differences in the amount of system workload for service advertisement. The best trade-off between discovery speed and system efficiency is once every 20 simulation steps in this example model.

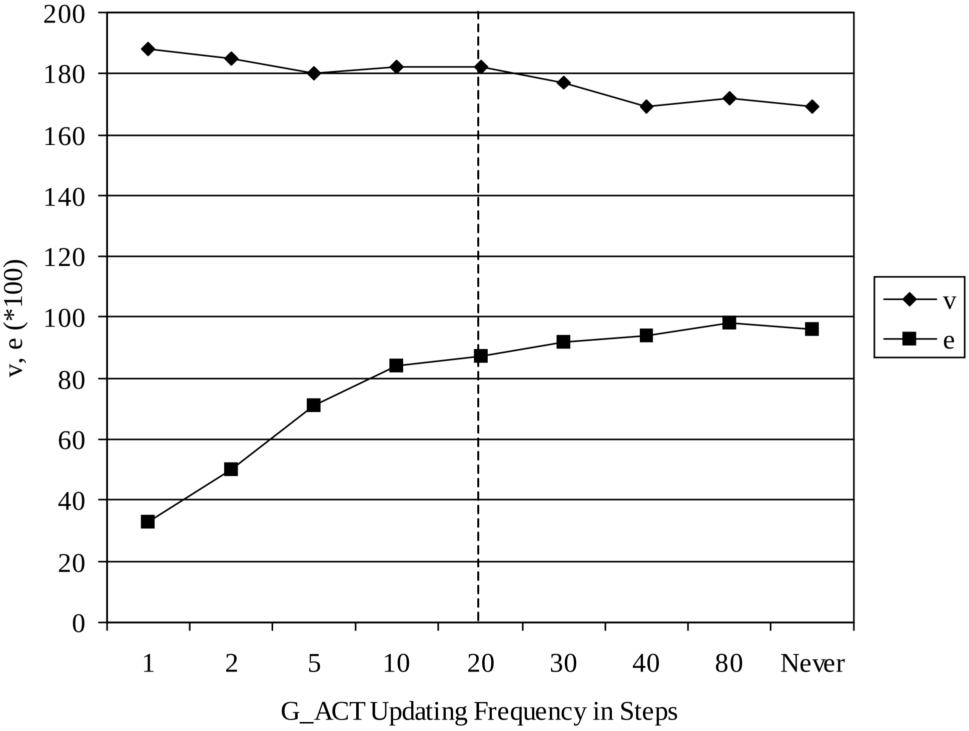


Figure 7.3 Choice of G_ACT Update Frequency

In summary, the example model should use all of the ACTs. L_ACT should be maintained by the real-time service advertisement. The G_ACT should be maintained by updating once every 20 steps. In fact, the performance of the example model can be improved further using agent level modelling. Different agents can use a mixture of different strategies to achieve higher performance of the whole system. This is not discussed in detail here.

The techniques of performance modelling and simulation are useful especially for the current phase of research into grid computing. As mentioned, a practical grid environment does not yet exist. In fact, there is not even a grid testbed that can be used for research. In the last chapter, the example system is composed of only 8 resources, which is far from a grid size. The performance data cannot be produced in such a system for analysis, which makes a simulation environment very valuable for this kind of research. The A4 simulator is such an attempt.

The PMA agent is used for online performance optimisation and steering for ARMS, which is a further usage of the simulation techniques. The simulation results are not only used for traditional performance analysis, but also feedback to the system for performance improvement in real time. However, the research into performance issues on service discovery in large-scale multi-agent system is just beginning. More performance optimisation strategies and steering policies need to be investigated further. A practical implementation of the ARMS and the PMA is ongoing, and is summarised in the conclusion part of the thesis.

Chapter 8

CONCLUSIONS

The grid is an emerging infrastructure for high performance computing. Resource management is the most important service for grid implementation. In this thesis, the methodology, tools, and applications of agent-based resource management for grid computing are presented. In this chapter, the main contents of the thesis are summarised and future work is suggested.

8.1 Thesis Summary

The work in this thesis is based on previous work on a performance evaluation toolkit, known as PACE. In this thesis, a new parallel application, Sweep3D, is used to validate the capabilities of performance modelling, evaluation, and prediction of the PACE system. The key features of PACE include rapid evaluation time, reasonable accuracy, and easy comparison across different platforms. The utilisation of PACE provides QoS support for grid resource management.

While extremely well suited for managing a locally distributed resource, the PACE functions do not map well onto a wide-area grid computing environment.

A new methodology for building large-scale distributed software systems with highly dynamic behaviours, A4 (Agile Architecture and Autonomous Agents), is presented in this thesis. The main component in an A4 system is the agent. Agents are both service requestors and providers. Services can be advertised and discovered within the hierarchy among different agents. There are four performance metrics for service discovery: discovery speed, system efficiency, load balancing, and success rate. A simulator for A4 has been developed that can be used for modelling and simulation to evaluate an A4 system performance.

The coupling of the A4 methodology with PACE functions leads to an initial implementation of an agent-based resource management system for grid computing, called ARMS. PACE is used to provide quantitative data concerning the performance of sophisticated applications running on a local resource. At a metacomputing level, agents cooperate with each other and perform resource advertisement and discovery functions to schedule applications that need to utilise the available resources. An ARMS agent includes: an ACT manager, the PACE evaluation engine, a multi-processor scheduler, and a matchmaker.

A special agent, a PMA, is also developed as a performance monitor and advisor in ARMS, which is capable of performance modelling and simulation of agent service discovery. Some performance steering policies can be used to guide the agents to choose different kinds of performance optimisation strategies, including the use of ACTs, limited service lifetime, and limited scope of service advertisement and discovery, etc, to improve system performance gradually.

The main contribution of this work includes: performance prediction driven QoS support for grid resource management and scheduling, an agent-based hierarchical model for service advertisement and discovery, and simulation-based performance optimisation and steering of agent resource discovery.

The performance prediction capability provided by the PACE toolkit was used for multi-processor scheduling, on-the-fly application steering, and traditional performance analysis. In the work described in this thesis, it is first used for QoS support of grid resource management. Most of the previous solutions to grid resource management include only soft QoS support. The key features of PACE make it a more suitable toolkit than any other evaluation tools to provide detail performance data rapidly without sacrificing the accuracy. This can be used to provide the hard QoS support for grid resource management at a meta level. The introduction of the PACE performance prediction technique to grid resource management differentiates this work from any other existing solutions.

Agent technologies have been developing for more than ten years and are becoming a mainstream software development technology. The development of the grid software infrastructure can benefit from the trend of agent-based software engineering in different ways. In this work, a hierarchy of homogenous agents is used with capabilities of service advertisement and discovery to provide grid resource management and scheduling at a meta level. Agents can be configured with different behaviours, which provides a flexible way for the system to adapt to the highly dynamic grid environment. The agent-based architecture not only provides a clean and powerful high-level abstraction of the grid resource management system described in this work, but can also be used as a framework for new components or functions to be added into the system. ARMS is the first prototype implementation of an agent-based resource management system for grid computing with important features that do not exist in other solutions.

Unlike many other agent-based system implementations that focus mainly on data representation and communication protocols, performance issues are the key consideration in the development of ARMS described in this thesis. The high-level performance evaluation and optimisation of service advertisement and discovery in large-scale MAS are attempted in this work using performance modelling and simulation techniques. Some performance metrics are defined and some performance optimisation strategies and steering policies are explored. Though performance issues on service discovery have been discussed in some other work, to the authors' knowledge, a quantitative analysis, that enables a MAS performance of service discovery to be investigated, can be only found in the work described in this thesis.

In summary, all of above go together to provides an available methodology and prototype implementation of an agent-based resource management system for grid computing, which can be used as a fundamental framework for further improvement and refinement.

8.2 Future Work

The main suggestion for future work is centred on the enhancement of ARMS. The framework and methodology have been demonstrated using an initial implementation of ARMS as described in this thesis. Many features can be added to the new implementation.

8.2.1 Performance Evaluation

The PACE toolkit is used to supply performance evaluation data in ARMS. There are still several aspects that can be improved for PACE to provide better QoS support of grid resource management.

Current PACE models include too much detail of an application or a resource, which need to be lightened for remote performance evaluation without sacrificing accuracy. A new project is to focus on transaction-level performance evaluation of Java applications [Spooner2001]. The detail of the operations in an application can be encapsulated into transactions, and the performance specification can be processed at a higher level. PACE models with lightweight application characterisations will reduce the communication workload between agents when service advertisement and discovery are processed, and hence improve the system performance.

PACE resource models are currently static without consideration of dynamic CPU workload and network traffic. The benchmark programs are executed off-line to produce these models on different platforms. In future, Dynamic Performance Measurement (DPM) can be applied to the ARMS implementation. The agents in

ARMS can control the benchmark programs to be executed on the local resource in real time and produce the corresponding resource models dynamically.

In the work described in this thesis, we focus on the evaluation of the application execution time, which is the only cost metrics that is included in the cost model of a request. In fact, more metrics (e.g. memory usage, execution environment, etc.) can be added into the cost model and the corresponding evaluation engines can also be added into each ARMS agent. This will provide a wider QoS support of the grid resource management.

8.2.2 Multi-processor Scheduling

An advanced multi-processor scheduling algorithm should be developed to include more consideration of dynamic information on resources and applications and aim to both meet requirements from users and maximise the resource utilisation.

A multi-processor scheduler, called TITAN, is under development at Warwick. A Genetic Algorithm (GA) is used as the kernel of TITAN. A monitoring module is also developed to collect dynamic information of the local processors. TITAN also takes advantage of the performance prediction capability of PACE. The GA in TITAN is an iterative heuristic process that can absorb slight changes of both resources and applications. TITAN aims to maximise the resource utilisation via calculating the penalty of the weighted idle time of the local processors and minimising the global make span of the application executions. An extension of the GA will aim to meet requirements from users as well.

TITAN will be an ideal local resource manager in the grid computing environment. The new implementation of ARMS can integrate multiple TITANs with agents to achieve grid resource management. TITAN can also be developed using the APIs provided by standard grid toolkits like Globus so as to cooperate with other kinds of local resource managers (e.g. Condor and AppLeS) in the grid environment.

8.2.3 Agent-based Resource Management

The A4 methodology and the ARMS agents can also be improved in a number of ways. These are listed in detail below.

- An agent in the hierarchy may be permitted to register with multiple upper agents, which will result in a more flexible and robust system architecture. Once an agent leaves the system, its lower agents are still able to contact the rest of the system via other upper agents. The cost of this would be more complex system management.
- New performance metrics for the agent-based service discovery can be developed concerning the communication time spent on the service discovery, instead of just the number of connections made for the service discovery. Benchmark programs can be developed to measure the communication time between two agents, and measurement results can be used for modelling the time spent in agent communication.
- New performance optimisation strategies and steering policies should be developed for efficient implementation of service advertisement and discovery in ARMS. The modelling and simulation techniques can be used to evaluate different strategies and their impact on the system performance
- New protocols for service advertisement and discovery can be developed to provide stronger QoS support. For example, multiple service support will provide users and system management tools with a wider base of QoS support. The agent-based grid resource discovery can also be designed to be a negotiation process between the users and the ARMS agents.
- Current agent behaviours in ARMS can only be configured by the system manager or the PMA. Further implementation of the ARMS agents should be able to change the behaviours themselves according to the changing requests and resources. The agent needs more capabilities to learn over time and get useful knowledge from its historic information.

8.2.4 Enhanced Implementation

The ARMS implementation can be enhanced using some existing standards, languages, tools and protocols. For example, the ARMS agents and the PMA can be developed using Java and an XML format for data representation. An agent communication language (ACL) can be used to allow agents to communicate with each other at a higher-abstracted knowledge level. A resource specification language (RSL) can be used to give a formal representation of service information in ARMS. Some network and database management protocols like LDAP and SNMP can also be used in the implementation of ARMS.

The new implementation of ARMS is to be tested on a grid infrastructure that is being built at Warwick. This includes clusters of Sun workstations, an SGI Origin2000 and an IBM S/390, etc. All of the work introduced above will enhance the applicability and usefulness of the implementation of ARMS towards a practical system.

BIBLIOGRAPHY

Alkindi2001 A. M. Alkindi, D. J. Kerbyson, G. R. Nudd, and E. Papaefstathiou, “Optimisation of Application Execution on Dynamic Systems”, Future Generation Computer Systems, Vol. 17, No. 8, Elsevier Science, pp. 941-949, 2001.

Amold1999 K. Amold, B. O’Sullivan, R. Scheifer, J. Waldo, and A. Woolrath, The Jini™ Specification, Addison Wesley, 1999.

Arbab1993 F. Arbab, I. Herman, and P. Spilling, “An Overview of Manifold and its Implementation”, Concurrency: Practice and Experience, Vol. 5, No. 1, pp. 23-70, 1993.

Atkins1996 D. E. Atkins, W. P. Birmingham, E. H. Durfee, E. J. Glover, T. Mullen, E. A. Rundensteiner, E. Soloway, J. M. Vidal, R. Wallace, and M. P. Wellman, “Toward Inquiry-Based Education Through Interacting Software Agents”, IEEE Computer, Vol. 29, No. 5, pp. 69-76, 1996.

Bagrodia1998 R. Bagrodia, R. Meyer, M. Takai, Y.A Chen, X. Zeng, J. Martin, and H.Y. Song, “Parsec: A Parallel Simulation Environment for Complex Systems”, IEEE Computer, Vol. 31, No. 10, pp. 77-85, 1998.

Baker2001 M. Baker, R. Buyya, and D. Laforenza, “The Grid: A Survey on Global Efforts in Grid Computing”, Technical Report: 2001/92, Monash University, Australia, May 2001.

Berman1996 F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, “Application-level Scheduling on Distributed Heterogeneous Networks”, in Proc. of Supercomputing, 1996.

Boloni1999 L. Boloni, and D. Marinescu, “An Object-Oriented Framework for Building Collaborative Network Agents”, in A. Kandel et al, eds, Agents in Intelligent Systems and Interfaces, Kluewer, 1999.

Bradshaw1997 J. M. Bradshaw, ed., Software Agents, The AAAI Press / The MIT Press, 1997.

Bray2000 J. Bray, and C. Sturman, Bluetooth: Connect Without Cables, Prentice Hall, 2000.

Brewington1999 B. Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko, and D. Rus, “Mobile Agents for Distributed Information Retrieval”, in M. Klusch, ed., Intelligent Information Agents, Chapter 15, Springer-Verlag, 1999.

Brooks1997 C. Brooks, B. Tierney, and W. Johnston, “JAVA Agents for Distributed System Management”, LBNL Report, 1997.

Buyya1999 R. Buyya, ed., “High Performance Cluster Computing”, Prentice Hall, 1999.

Buyya2000 R. Buyya, D. Abramson, and J. Giddy, “Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid”, in Proc. Of 4th Int. Conf. on High Performance Computing in Asia-Pacific Region, Beijing, China, 2000.

Buyya2000b R. Buyya, S. Chapin, and D. DiNucci, “Architectural Models for Resource Management in the Grid”, in Proc. of 1st IEEE/ACM Int. Workshop on Grid Computing, Lecture Notes in Computer Science 1971, Springer-Verlag, pp. 18-35, 2000.

Cao1996 J. Cao, “Analysis and Design of an Acknowledge Subsystem of BMCST-MIS”, BEng Thesis, Tsinghua University, 1996.

Cao1998 J. Cao, “Flexible Software Systems”, MSc Thesis, Tsinghua University, 1998.

Cao1999 J. Cao, Y. Fan, and C. Wu, “Research of Operation Administration System Agents of Integration Platform”, Computer Integrated Manufacturing Systems (CIMS), Vol. 5, No. 3, pp. 39-43, 1999.

Cao1999b J. Cao, Y. Fan, and C. Wu, “System Architecture of New CIMS Application Integration Platform”, J. of Tsinghua University, Vol. 39, No. 7, pp. 68-71, 1999.

Cao1999c J. Cao, and Y. Fan, “Concepts of Flexible Software Systems”, Computer Science, Vol. 26, No. 2, pp. 74-77, 1999.

Cao1999d J. Cao, D. J. Kerbyson, E. Papaefstathiou, and G. R. Nudd, “Modeling of ASCI High Performance Applications Using PACE”, in Proc. of 15th Annual UK Performance Engineering Workshop, Bristol, UK, pp. 413-424, 1999.

Cao2000 J. Cao, D. J. Kerbyson, E. Papaefstathiou, and G. R. Nudd, “Performance Modelling of Parallel and Distributed Computing Using PACE”, in Proc. of 19th IEEE Int. Performance, Computing and Communication Conf., Phoenix, USA, pp. 485-492, 2000.

Cao2000b J. Cao, D. J. Kerbyson, and G. R. Nudd, “Dynamic Application Integration Using Agent-Based Operational Administration”, in Proc. of 5th Int. Conf. on Practical Application of Intelligent Agents and Multi-Agent Technology, Manchester, UK, pp. 393-396, 2000.

Cao2001 J. Cao, D. J. Kerbyson, and G. R. Nudd, “Performance Evaluation of an Agent-Based Resource Management Infrastructure for Grid Computing”, in Proc. of 1st IEEE Int. Symp. on Cluster Computing and the Grid, Brisbane, Australia, pp. 311-318, 2001.

Cao2001b J. Cao, D. J. Kerbyson, and G. R. Nudd, “Use of Agent-Based Service Discovery for Resource Management in Metacomputing Environment”, in Proc. of 7th Int. Euro-Par Conf., Manchester, UK, Lecture Notes in Computer Science 2150, Springer-Verlag, pp. 882-886, 2001.

Cao2001c J. Cao, D. J. Kerbyson, and G. R. Nudd, “High Performance Service Discovery in Large-scale Multi-agent and Mobile-agent Systems”, to appear in Int. J. of Software Engineering and Knowledge Engineering, Special Issue on Muti-Agent Systems and Mobile Agents, World Scientific Publishing, 2001.

Cao2001d J. Cao, D. J. Kerbyson, S. A. Jarvis, G. R. Nudd, D. P. Spooner, and J. D. Turner, “ARMS: an Agent-based Resource Management System for Grid Computing”, to appear in Scientific Programming, Special Issue on Grid Computing, IOS Press, 2001.

Carriero1989 N. Carriero, and D. Gelernter, “Linda in Context”, Communications of ACM, Vol. 32, No. 4, pp. 444-458, 1989.

Casanova1998 H. Casanova, and J. Dongarra, “Applying NetSolve’s Network-Enabled Server”, IEEE Computational Science & Engineering, Vol. 5, No. 3, pp. 57-67, 1998.

Case1988 J. Case, M. Fedor, M. Schoffstall, and J. Davin, “A Simple Network Management Protocol”, RFC 1067, IETF Draft Standard, 1988.

Chapin1999 S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw, “Resource Management in Legion”, Future Generation Computer Systems, Vol. 15, No. 5, pp. 583-594, 1999.

Chen1996 H. Chen, A. Houston, J. Nunamaker, and J. Yen, “Toward Intelligent Meeting Agents”, IEEE Computer, Vol. 29, No. 8, pp. 62-70, 1996.

Ciancarini1996 P. Ciancarini, “Coordination Models and Languages as Software Integrators”, ACM Computing Surveys, Vol. 28, No. 2, pp. 300-302, 1996.

Ciancarini1999 P. Ciancarini, and A. L. Wolf (eds.), Proc. of 3rd Int. Conf. on Coordination Languages and Models, Lecture Notes on Computer Science 1594, Springer Verlag, 1999.

Ciancarini2001 P. Ciancarini, and M. Wooldridge (eds.), Agent-Oriented Software Engineering, Lecture Notes in Artificial Intelligence, Vol. 1957, Springer Verlag, 2001.

Czajkowski1998 K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, “A Resource Management Architecture for Metacomputing Systems”, in Proc. of IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

Davison1998 R. G. Davison, J. J. Hardwicke, and M. D. J. Cox, “Applying the Agent Paradigm to Network Management”, BT Technology Journal, Vol.16, No. 3, pp. 86-93, July 1998.

Denis2001 A. Denis, C. Pérez, and T. Priol, "Portable Parallel CORBA Objects: An Approach to Combine Parallel and Distributed Programming for Grid Computing", in Proc. of 7th Int. Euro-Par Conf., Manchester, UK, Lecture Notes in Computer Science 2150, Springer-Verlag, pp. 835-844, 2001.

Deelman1998 E. Deelman, A. Dube, A. Hoisie, Y. Luo, R. L. Oliver, D. Sundaram-Stukel, H. Wasserman, V. S. Adve, R. Bagrodia, J. C. Browne, E. Houstis, O. Lubeck, J. Rice, P. J. Teller, and M. K. Vernon, "POEMS: End-to-end Performance Design of Large Parallel Adaptive Computational Systems", in Proc. of the 1st ACM Int. Workshop on Software and Performance, pp. 18-30, 1998.

Dongarra1994 J. Dongarra, D. Walker, E. Lusk, et al., "Special Issue - MPI - A Message-Passing Interface Standard", Int. J. of Supercomputer Applications and High Performance Computing, Vol. 8, No. 3-4, 1994.

Fan1999 Y. Fan, W. Shi, and C. Wu, "Enterprise Wide Application Integration Platform for CIMS Implementation", J. of Intelligent Manufacturing, Vol. 10, No. 6, pp. 587-601, 1999.

Fan2000 Y. Fan, and J. Cao, Object-oriented Modelling, Analysis and Design of Complex Systems, Tsinghua University Press / Springer-Verlag, 2000.

Fan2001 Y. Fan, and J. Cao, Multi-Agent Systems: Theories, Applications and Methods, to be published by Tsinghua University Press / Springer-Verlag, 2001.

Fitzgerald1997 S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations", in Proc. of 6th IEEE Symp. on High Performance Distributed Computing, pp. 365-375, 1997.

Foster1997 I. Foster, and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", Int. J. Supercomputer Applications, Vol. 11, No. 2, pp. 115-128, 1997.

Foster1998 I. Foster, and C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, 1998.

Foster1999 I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation", in Proc. of Int. Workshop on Quality of Service, 1999.

Foster2000 I. Foster, "Internet Computing and the Emerging Grid", Nature, Dec. 7, 2000.

Foster2001 I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", to be published in Intl. J. Supercomputer Applications, 2001.

Frank1997 M. I. Frank, A. Agarwal, and M. K. Vernon, "LoPC: Modelling Contention in Parallel Algorithms", in Proc. of 6th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming, Las Vegas, pp. 62-73, 1997.

Furmento2001 N. Furmento, S. Newhouse, and J. Darlington, "Building Computational Communities from Federated Resources", in Proc. of 7th Int. Euro-Par Conf., Manchester, UK, Lecture Notes in Computer Science 2150, Springer-Verlag, pp. 855-863, 2001.

Garlan1993 D. Garlan, and M. Shaw, "An Introduction to Software Achitecture", in Advances in Software Engineering and Knowledge Engineering, World Scientific, Vol. 1, 1993.

Geist1994 A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing, MIT Press, 1994.

Gelernter1992 D. Gelernter, and N. Carriero, "Coordination Languages and Their Significance", Communications of the ACM, Vol. 35, No. 2, pp. 96-107, 1992.

Goland1999 Y. Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright, "Simple Service Discovery Protocol/1.0: Operating without an Arbiter", IETF Internet Draft, 1999.

Grimshaw1999 A. Grimshaw, A. Ferrari, F. Knabe, and M. Humphrey, "Wide-Area Computing: Resource Sharing on a Large Scale", IEEE Computer, Vol. 32, No. 5, pp. 29-37, 1999.

Guttman1999 E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2", RFC 2608, IETF Draft Standard, 1998.

Hall1996 M. W. Hall, J. M. Anderson, S. P. Amarasinghe, B. R. Murphy, S. Liao, E. Bugnion, and M. S. Lam, "Maximizing Multiprocessor Performance with the SUIF Compiler", IEEE Computer, Vol. 29, No. 12, pp. 84-89, 1996.

Harper1999 J. Harper, D. J. Kerbyson, and G. R. Nudd, "Analytical Modeling of Set-Associative Cache Behavior", IEEE Trans. on Computers, Vol. 48, No. 10, pp. 1009-1024, 1999.

Hey2001 T. Hey, "e-Science Core Programme", in e-Science Core Programme Town Meeting, London, UK, 2001.

Jennings1998 N. R. Jennings, and M. J. Wooldridge (eds), Agent Technology: Foundations, Applications, and Markets, Springer-Verlag, 1998.

Jennings2000 N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, and B. Odgers, "Autonomous Agents for Business Process Management", Int. J. of Applied Artificial Intelligence, Vol. 14, No. 2, pp. 145-189, 2000.

Jennings2000b N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, B. Odgers, and J. L. Alty, "Implementing a Business Process Management System using ADEPT: A Real-World Case Study", Int. J. of Applied Artificial Intelligence, Vol. 14, No. 5, pp. 421-465, 2000.

Jennings2001 N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge, "Automated Negotiation: Prospects, Methods and Challenges", Int. J. of Group Decision and Negotiation, Vol. 10, No. 2, pp. 199-215, 2001.

Jennings2001b N. R. Jennings, "An Agent-based Approach for Building Complex Software Systems", Communications of the ACM, Vol. 44, No. 4, pp. 35-41, 2001.

Jeon2000 H. Jeon, C. Petrie, and M. R. Cutkosky, "JATLite: A Java Agent Infrastructure with Message Routing", IEEE Internet Computing, Vol. 4, No. 2, pp. 87-96, 2000.

Jini1999 "Jini™ Architectural Overview", Sun Technical White Paper, Jan. 1999.

Kerbyson1998 D. J. Kerbyson, E. Papaefstathiou, and G. R. Nudd, "Application Execution Steering Using On-the-fly Performance Prediction", in Proc. of Euro. Conf. on High Performance Computing and Networking, Lecture Notes in Computer Science 1401, Springer-Verlag, pp. 718-727, 1998.

Kerbyson2000 D. J. Kerbyson, J. S. Harper, E. Papaefstathiou, D. V. Wilcox, and G. R. Nudd, "Use of Performance Technology for the Management of Distributed Systems", in Proc. of 6th Int. Euro-Par Conf., Lecture Notes in Computer Science 1900, Springer-Verlag, pp. 149-159, 2000.

Koch1992 K. R. Koch, R. S. Baker, and R. E. Alcouffe, "Solution of the First-Order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor", Trans. of the Amer. Nuc. Soc., Vol. 65, No. 108, 1992.

Kon2000 F. Kon, R. Campbell, M. Mickunas, and K. Nahrstedt, "2K: A Distributed Operating System for Dynamic Heterogeneous Environments", in Proc. of 9th IEEE Int. Symp. on High Performance Distributed Computing, 2000.

Kraus1998 S. Kraus, K. Sycara, and A. Evenchik, "Reaching Agreements through Argumentation: a Logical Model and Implementation", Artificial Intelligence, Vol. 104, pp. 1-69, 1998.

Krauter2000 K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems", Technical Report: 2000/80, Monash University, Australia, Nov. 2000.

Labrou1999 Y. Labrou, T. Finin, and Y. Peng, "Agent Communication Languages: The Current Landscape", IEEE Intelligent Systems, Vol. 14, No. 2, pp. 45-52, 1999.

Lea2001 R. Lea, S. Gibbs, R. Gauba, and R. Balaraman, HAVi Example By Example: Java Programming for Home Entertainment Devices, Prentice Hall, 2001.

Leinberger1999 W. Leinberger, and V. Kumar, "Information Power Grid: The New Frontier in Parallel Computing", IEEE Concurrency, Vol. 7, No. 4, pp. 75-84, 1999.

Lieberman1997 H. Lieberman, "Autonomous Interface Agents", in CHI '97 Conf. Proc. on Human Factors in Computing Systems, pp. 67-74, 1997.

Litzkow1988 M. Litzkow, M. Livny, and Matt Mutka, “Condor - A Hunter of Idle Workstations”, in Proc. of 8th Int. Conf. on Distributed Computing Systems, pp. 104-111, June 1988.

Maes1995 P. Maes, “Artificial Life Meets Entertainment: Lifelike Autonomous Agents”, Communications of the ACM, Vol, 38, No. 11, pp. 108-114, 1995.

Malone1994 T. W. Malone, and K. Crowston, “The Interdisciplinary Study of Coordination”, ACM Computing Survey, Vol. 26, No. 1, pp. 87-119, 1994.

Miller1995 B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall, “The Paradyn Parallel Performance Measurement Tools”, IEEE Computer, Vol. 28, No. 11, pp. 37-46, 1995.

Miller1999 B. Miller, and R. Pascoe, “Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer”, Bluetooth White Paper, July 1999.

Nakada1998 H. Nakada, H. Takagi, S. Matsuoka, U. Nagashima, M. Sato, and S. Sekiguchi, “Utilizing the Metaserver Architecture in the Ninf Global Computing System”, in Proc. of High Performance Computing and Networking, Lecture Notes on Computer Science 1401, Springer-Verlag, pp. 607-616, 1998.

Nowak1997 D. A. Nowak, R. C. Christensen, “ASCI Applications”, LLNL Report 232247, Nov. 1997.

Nudd2000 G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox, “PACE – A Toolset for the Performance Prediction of Parallel and Distributed Systems”, Int. J. of High Performance Computing Applications, Special Issues on Performance Modelling – Part I, Sage Science Press, Vol. 14, No. 3, pp. 228-251, Fall 2000.

Nwana1998 H. S. Nwana, J. Rosenschein, T. Sandholm, C. Sierra, P. Maes, and R. Guttmann, “Agent-Mediated Electronic Commerce: Issues, Challenges and Some Viewpoints” in Proc. of 2nd ACM Int. Conf. on Autonomous Agents, pp. 189-196, 1998.

Papadopoulos1998 G. Papadopoulos, and F. Arbab, “Coordination Models and Languages”, in Advances in Computers, Vol. 46: The Engineering of Large Systems, Academic Press, 1998.

Papaefstathiou1994 E. Papaefstathiou, D. J. Kerbyson, G. R. Nudd, and T. J. Atherton, “An Analysis of Processor Resource Models for Use in Performance Prediction”, Research Report CS-RR-279, Department of Computer Science, University of Warwick, 1994.

Papaefstathiou1995 E. Papaefstathiou, D. J. Kerbyson, G. R. Nudd, and T. J. Atherton, “An Overview of the CHIP^{3}S Performance Prediction Toolset for Parallel Systems”, in Proc. of 8th ISCA Int. Conf. on Parallel and Distributed Computing Systems, pp. 527-533, 1995.

Papaefstathiou1995b E. Papaefstathiou, “A Framework for Characterising Parallel Systems for Performance Evaluation”, Ph.D. Thesis, Department of Computer Science, University of Warwick, 1995.

Papaefstathiou1998 E. Papaefstathiou, D. J. Kerbyson, G.R. Nudd, J. S. Harper, S. C. Perry, and D. V. Wilcox, “A Performance Analysis Environment for Life”, in Proc. of 2nd ACM SIGMETRICS Symp. on Parallel and Distributed Tools, Oregon, USA, 1998.

Parunak1998 H. V. D. Parunak, “Practical and Industrial Applications of DAI”, in G. Weiss (ed.), Introduction to Distributed Artificial Intelligence, MIT Press, 1998.

Parunak2001 H. V. D. Parunak, A. D. Baker, and S. J. Clark, “AARIA Agent Architecture: from Manufacturing Requirements to Agent-based System Design”, J. of Integrated Computer-Aided Engineering, Vol. 8, No. 1, pp. 45-58, 2001.

Pascoe2001 R. Pascoe, “Building Networks on the Fly”, IEEE Spectrum, Vol. 38, No. 3, pp. 61-65, 2001.

Perry1992 D. E. Perry, and A. L. Wolf, “Foundations for the Studies of Software Architecture”, ACM SIGSOFT Software Engineering Notes, Vol. 17, No. 4, 1992.

Perry1999 S. C. Perry, J. S. Harper, D. J. Kerbyson, and G. R. Nudd, “Theory and Operation of the Warwick Multiprocessor Scheduling System”, Research Report CS-RR-363, Dept. of Computer Science, University of Warwick, 1999.

Perry2000 S. C. Perry, R. H. Grimwood, D. J. Kerbyson, E. Papaefstathiou, and G. R. Nudd, "Performance Optimisation of Financial Option Calculations", Parallel Computing, Vol. 26, No. 5, pp. 623-639, 2000.

Qin1991 B. Qin, H. A. Sholl, and R. A. Ammar, "Micro Time Cost Analysis of Parallel Computations", IEEE Trans. on Computers, Vol. 40, No. 5, pp. 613-628, 1991.

Raman1998 R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", in Proc. of 7th IEEE Int. Symp. on High Performance Distributed Computing, Chicago, USA, July 1998.

Rana2000 O. F. Rana, and D. W. Walker, "The Agent Grid: Agent-Based Resource Integration in PSEs", in Proc. of 16th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation, Lausanne, Switzerland, 2000.

Richard2000 G. G. Richard III, "Service Advertisement and Discovery: Enabling Universal Device Cooperation", IEEE Internet Computing, Vol. 4, No. 5, pp. 18-26, 2000.

Sato1998 M. Sato, H. Tezuka, A. Hori, Y. Ishikawa, S. Sekiguchi, H. Nakada, S. Matsuoka, and U. Nagashima, "Ninf and PM: Communication Libraries for Global Computing and High-performance Cluster Computing", Future Generation Computer Systems, Vol. 13, No. 4-5, pp. 349-359, 1998.

Schopf1997 J. M. Schopf, "Structural Prediction Models for High-Performance Distributed Applications", in Proc. of 1997 Cluster Computing Conf., 1997.

Segal2000 B. Segal, "Grid Computing: The European Data Grid Project", in Proc. of IEEE Nuclear Science Symp. and Medical Imaging Conf., Lyon, France, 2000.

Sevilla2001 D. Sevilla, J. M. García, and A. Gómez, "CORBA Lightweight Components: A Model for Distributed Component-Based Heterogeneous Computation", in Proc. of 7th Int. Euro-Par Conf., Manchester, UK, Lecture Notes in Computer Science 2150, Springer-Verlag, pp. 845-854, 2001.

Singh1998 M. P. Singh, “Agent Communication Languages: Rethinking the Principles”, IEEE Computer, Vol. 31, No. 12, pp. 40-47, 1998.

Slama1999 D. Slama, J. Garbis, and P. Russell, Enterprise Corba, Prentice Hall, 1999.

Smith1990 C. U. Smith, Performance Engineering of Software Systems, Addison Wesley, 1990.

Spooner2001 D. P. Spooner, J. D. Turner, J. Cao, S. A. Jarvis, and G. R. Nudd, “Application Characterisation Using a Lightweight Transaction Model”, in Proc. of 17th Annual UK Performance Engineering Workshop, Leeds, UK, pp. 215-225, 2001.

Tecuci1998 G. Tecuci, Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies, Academic Press, 1998.

Tierney2000 B. Tierney, W. Johnston, J. Lee, and M. Thompson, “A Data Intensive Distributed Computing Architecture for Grid Applications”, Future Generation Computer Systems, Vol. 16, No. 5, pp 473-481, 2000.

Tierney2001 B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, and M. Swany, “A Grid Monitoring Service Architecture”, White Paper, Grid Performance Working Group, Global Grid Forum, 2001.

UPnP2000 “Understanding Universal Plug and Play”, Microsoft White Paper, Jun. 2000.

Uysal1998 M. Uysal, T. M. Kurc, A. Sussman, and J. Saltz, “A Performance Prediction Framework for Data Intensive Applications on Large Scale Parallel Machines”, in Proc. of 4th Workshop on Languages, Compilers and Run-time Systems for Scalable Computers, 1998.

Wolfram1991 S. Wolfram, Mathematica: a System for Doing Mathematics by Computer, Second Edition, Addison Wesley, 1991.

Wooldridge1999 M. J. Wooldridge, and N. R. Jennings, “Software Engineering with Agents: Pitfalls and Pratfalls”, IEEE Internet Computing, Vol. 3, No. 3, pp. 20-27, 1999.

Yeong1995 W. Yeong, T. Howes, and S. Kille, “Lightweight Directory Access Protocol”, RFC 1777, IETF Draft Standard, 1995.

Appendix A

PSL CODE FOR SWEEP3D

This section gives a list of all the PSL source code for Sweep3D. Each software object is included in a separate section. There are totally 9 objects: 1 application object, 4 subtask objects, and 4 parallel template objects.

A.1 Application Object: sweep3d

```
(*
 * SWEEP3D model
 *)

application sweep3d {

        include hardware;

        include source;
        include sweep;
        include fixed;
        include flux_err;

        var numeric:
                Nproc = 6,
                npe_i = 2,
                npe_j = 3,
                mk = 10,
                mmi = 3,
                it_g = 50,
                jt_g = 50,
                kt = 50,
                mm = 6,
                isct = 1,
                epsi = -12,
                ibc = 0,
                jbc = 0,
                kbc = 0,
                do_dsa = 1,
                ifixups = -7,
                it,
                jt,
                it_dsa,
                jt_dsa,
                kt_dsa,
                jt_ibc,
                kt_ibc,
                mm_ibc,
                it_jbc,
                kt_jbc,
                mm_jbc,
                it_kbc,
                jt_kbc,
                mm_kbc,
                nk,
                ndiag,
                nm;


        link {
        hardware:
                Nproc = Nproc;
        source:
                it = it,
                jt = jt,
                kt = kt,
                isct = isct,
                ifixups = ifixups,
                epsi = epsi;
        sweep:
                it = it,
                jt = jt,
                kt = kt,
                do_dsa = do_dsa,
                it_dsa = it_dsa,
                jt_dsa = jt_dsa,
                kt_dsa = kt_dsa,
                ibc = ibc,
                jbc = jbc,
                kbc = kbc,
                mm = mm,
                mmi = mmi,
                nk = nk,
                mk = mk,
                ndiag = ndiag,
                nm = nm,
                epsi = epsi,
                ifixups = ifixups,
                npe_i = npe_i,
                npe_j = npe_j;

        flux_err:
                it = it,
                jt = jt,
                kt = kt;
        }

        option {
                hrduse = "SunUltra1";
        }

        proc exec init {
                var numeric:
                        i,tmp;

                if (Nproc == 1)
                {
                        npe_i = 1;
                        npe_j = 1;
                }
                else if (Nproc == 2)
                {
                        npe_i = 1;
                        npe_j = 2;
                }
                else if (Nproc == 3)
                {
                        npe_i = 1;
                        npe_j = 3;
                }
                else if (Nproc == 4)
                {
                        npe_i = 2;
                        npe_j = 2;
                }
                else if (Nproc == 5)
                {
                        npe_i = 1;
                        npe_j = 5;
                }
                else if (Nproc == 6)
                {
                        npe_i = 2;
                        npe_j = 3;
                }
                else if (Nproc == 7)
                {
                        npe_i = 1;
                        npe_j = 7;
                }
                else if (Nproc == 8)
                {
                        npe_i = 2;
                        npe_j = 4;
                }
                else if (Nproc == 9)
                {
                        npe_i = 3;
                        npe_j = 3;
                }
                else if (Nproc == 10)
                {
                        npe_i = 2;
                        npe_j = 5;
                }
                else if (Nproc == 11)
                {
                        npe_i = 1;
                        npe_j = 11;
                }
                else if (Nproc == 12)
                {
                        npe_i = 3;
                        npe_j = 4;
                }
                else if (Nproc == 13)
                {
                        npe_i = 1;
                        npe_j = 13;
                }
                else if (Nproc == 14)
                {
                        npe_i = 2;
                        npe_j = 7;
                }
                else if (Nproc == 15)
                {
                        npe_i = 3;
                        npe_j = 5;
                }
                else if (Nproc == 16)
                {
                        npe_i = 4;
                        npe_j = 4;
                }

                if (isct == 0) nm=1;
                else if (isct == 1) nm=4;

                it = it_g / npe_i ;
                jt = jt_g / npe_j + 1 ;
```

```
                if( mk > kt ) mk = kt;

                if (do_dsa == 1)
                {
                        it_dsa = it + 1;
                        jt_dsa = jt + 1;
                        kt_dsa = kt + 1;
                }
                else
                {
                        it_dsa = 1;
                        jt_dsa = 1;
                        kt_dsa = 1;
                }

                if (ibc != 0)
                {
                        jt_ibc = jt;
                        kt_ibc = kt;
                        mm_ibc = mm;
                }
                else
                {
                        jt_ibc = 1;
                        kt_ibc = 1;
                        mm_ibc = 1;
                }

                if (jbc!=0)
                {
                        it_jbc = it;
                        kt_jbc = kt;
                        mm_jbc = mm;
                }
                else
                {
                        it_jbc = 1;
                        kt_jbc = 1;
                        mm_jbc = 1;
                }

                if (kbc != 0)
                {
                        it_kbc = it;
                        jt_kbc = jt;
                        mm_kbc = mm;
                }
                else
                {
                        it_kbc = 1;
                        jt_kbc = 1;
                        mm_kbc = 1;
                }

                tmp = kt;
                i = 1;
                while ( tmp > mk )
                {
                        tmp = tmp - mk;
                        i = i + 1;
                }

                nk = kt / i;
                ndiag = (nk+jt+i+mmi)*jt / (nk+jt);

                for( i = 1; i <= -epsi; i = i + 1 )
                {
                        call source;
                        call sweep;
                        call fixed;
                        call flux_err;
                }
        }
}
```

A.2 Subtask Object: source

```
subtask source {
        include async;
        include hardware;

        var numeric:
                it = 25,
                jt = 17,
                kt = 50,
                isct = 1,
                ifixups = -7,
                epsi = -12,
                p1,
                p2;

        link {
                async: Tx = source_comp();
        }

        proc exec init {
                if( ifixups > 0 )
                {
                        p1 = 1;
                        p2 = 0;
                }
                if( ifixups == 0 )
                {
                        p1 = 0;
                        p2 = 1;
                }
                if( ifixups < 0 )
                {
                        p1 = 0;
                        p2 = 0;
                }

        }

(*
 * CHIP3S
 * Application Characterisation Tool
 * Source  : source.c
 * RUV Type: clc
 *)

    (* Calls: *)
    proc cflow source_comp {  (* Defined at source.c:1 *)
        compute <is clc, FCAL, 2*POL1, AILL, TILL, CMLL>;
        case (<is clc, IFBR>) {
        1-isct:
            compute <is clc, SILL>;
            loop (<is clc, LFOR>, kt) {
                compute <is clc, CMLL, SILL>;
                loop (<is clc, LFOR>, jt) {
                    compute <is clc, CMLL, SILL>;
                    loop (<is clc, LFOR>, it) {
                        compute <is clc, CMLL, 7*ARD3, MFDL, AFDL, 2*TFDL, SFDL
                            , INLL>;
                    }
                    compute <is clc, INLL>;
                }
                compute <is clc, INLL>;
            }
        1-(1-isct):
            compute <is clc, SILL>;
            loop (<is clc, LFOR>, kt) {
                compute <is clc, CMLL, SILL>;
                loop (<is clc, LFOR>, jt) {
                    compute <is clc, CMLL, SILL>;
                    loop (<is clc, LFOR>, it) {
                        compute <is clc, CMLL, 19*ARD3, 4*MFDL, AFDL, 5*TFDL
                            , 4*SFDL, INLL>;
                    }
                    compute <is clc, INLL>;
                }
                compute <is clc, INLL>;
            }
        }
        compute <is clc, SILL, CMLL>;
        case (<is clc, IFBR>) {
        p1:
            compute <is clc, SILL>;
        1-(p1):
            compute <is clc, CMLL>;
            case (<is clc, IFBR>) {
            p2:
                compute <is clc, SILL>;
            1-(p2):
                compute <is clc, SILL, POL1, CMLL>;
                case (<is clc, IFBR>) {
                (-epsi+ifixups)/(-epsi):
                    compute <is clc, SILL>;
                }
            }
        }
    } (* End of source_comp *)

}
```

A.3 Subtask Object: sweep

```
subtask sweep {
       include hardware;
       include pipeline;

       var numeric:
              it = 26,
              jt = 18,
              kt = 50,
              do_dsa = 1,
              mm = 6,
              mk = 10,
              mmi = 3,
              it_dsa = 26,
              jt_dsa = 18,
              kt_dsa = 51,
              ibc = 0,
              jbc = 0,
              kbc = 0,
              nk = 9,
              ndiag = 17,
              nm = 4,
              npe_i = 2,
              npe_j = 3,
              epsi = -12,
              ifixups = -7,
              p1,
              p2,
              p3;

       link {
       pipeline:
              Tx_sweep_init = sweep_init(),
              Tx_octant = octant(),
              Tx_get_direct = get_direct(),
              Tx_pipeline_init = pipeline_init(),
              Tx_kk_loop_init = kk_loop_init(),
              Tx_else_ew_rcv = else_ew_rcv(),
              Tx_comp_face = comp_face(),
              Tx_else_ns_rcv = else_ns_rcv(),
              Tx_work = work(),
              Tx_else_ew_snd = else_ew_snd(),
              Tx_else_ns_snd = else_ns_snd(),
              Tx_last = last(),
              mm = mm,
              mmi = mmi,
              it = it,
              jt = jt,
                kt = kt,
                mk = mk,
                npe_i = npe_i,
                npe_j = npe_j;
        }

        proc exec init {
                if(kbc == 0)
                        p1 = 1;
                else
                        p1 = 0.5;
                if(ibc == 0)
                        p2 = 1;
                else
                        p2 = 0.5;
                if(jbc == 0)
                        p3 = 1;
                else
                        p3 = 0.5;
        }

(*
 * CHIP3S
 * Application Characterisation Tool
 * Source  : sweep.c
 * RUV Type: clc
 *)

    (* Calls: *)
    proc cflow sweep_init {   (* Defined at sweep.c:2 *)
        compute <is clc, FCAL, 6*ARD1, 6*SFDL>;
        case (<is clc, IFBR>) {
        do_dsa:
            compute <is clc, SILL>;
            loop (<is clc, LFOR>, 3) {
                compute <is clc, CMLL, SILL>;
                loop (<is clc, LFOR>, kt_dsa) {
                    compute <is clc, CMLL, SILL>;
                    loop (<is clc, LFOR>, jt_dsa) {
                        compute <is clc, CMLL, SILL>;
                        loop (<is clc, LFOR>, it_dsa) {
                            compute <is clc, CMLL, ARD3, SFDL, INLL>;
                        }
                        compute <is clc, INLL>;
                    }
                    compute <is clc, INLL>;
                }
                compute <is clc, INLL>;
            }
        }
        compute <is clc, POL1, SILL>;
    } (* End of sweep_init *)

    (* Calls: *)
    proc cflow octant {       (* Defined at sweep.c:74 *)
        compute <is clc, FCAL, CMLL>;
        case (<is clc, IFBR>) {
        1/8:
            compute <is clc, 9*POL1, 6*TILL, 3*SILL>;
        1-(1/8):
            compute <is clc, CMLL>;
            case (<is clc, IFBR>) {
            1/7:
                compute <is clc, 9*POL1, 5*TILL, 4*SILL>;
            1-(1/7):
                compute <is clc, CMLL>;
                case (<is clc, IFBR>) {
                1/6:
                    compute <is clc, 9*POL1, 5*TILL, 4*SILL>;
                1-(1/6):
                    compute <is clc, CMLL>;
                    case (<is clc, IFBR>) {
                    1/5:
                        compute <is clc, 9*POL1, 4*TILL, 5*SILL>;
                    1-(1/5):
                        compute <is clc, CMLL>;
                        case (<is clc, IFBR>) {
```

```
                    1/4:
                        compute <is clc, 9*POL1, 4*SILL, 5*TILL>;
                    1-(1/4):
                        compute <is clc, CMLL>;
                        case (<is clc, IFBR>) {
                        1/3:
                            compute <is clc, 9*POL1, 5*SILL, 4*TILL>;
                        1-(1/3):
                            compute <is clc, CMLL>;
                            case (<is clc, IFBR>) {
                            1/2:
                                compute <is clc, 9*POL1, 5*SILL, 4*TILL>;
                            1-(1/2):
                                compute <is clc, 9*POL1, 6*SILL, 3*TILL>;
                            }
                        }
                    }
                }
            }
        }
    }
    compute <is clc, POL1, CMLL>;
    case (<is clc, IFBR>) {
    0.5:
        compute <is clc, POL1, SILL>;
    1-(0.5):
        compute <is clc, POL1, SILL>;
    }
    compute <is clc, POL1, CMLL>;
    case (<is clc, IFBR>) {
    0.5:
        compute <is clc, POL1, SILL>;
    1-(0.5):
        compute <is clc, POL1, SILL>;
    }
    compute <is clc, POL1, CMLL>;
    case (<is clc, IFBR>) {
    0.5:
        compute <is clc, POL1, SILL>;
    1-(0.5):
        compute <is clc, POL1, SILL>;
    }
} (* End of octant *)

(* Calls: *)
proc cflow get_direct {   (* Defined at sweep.c:202 *)
    compute <is clc, FCAL, CMLL>;
    case (<is clc, IFBR>) {
    0.5:
        compute <is clc, 3*TILL, MILL>;
    1-(0.5):
        compute <is clc, 3*TILL, MILL>;
    }
    compute <is clc, CMLL>;
    case (<is clc, IFBR>) {
    0.5:
        compute <is clc, 3*TILL, MILL>;
    1-(0.5):
        compute <is clc, 3*TILL, MILL>;
    }
    compute <is clc, SILL>;
    loop (<is clc, LFOR>, mm) {
        compute <is clc, CMLL, 9*ARD1, 6*MILL, 3*TFDL, INLL>;
    }
} (* End of get_direct *)

(* Calls: *)
proc cflow pipeline_init {       (* Defined at sweep.c:296 *)
    compute <is clc, FCAL, AILL, MILL, TILL, 2*CMLL, ANDL>;
    case (<is clc, IFBR>) {
    p1:
        compute <is clc, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, SILL>;
            loop (<is clc, LFOR>, jt) {
                compute <is clc, CMLL, SILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, CMLL, ARD3, SFDL, INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
    1-(p1):
        case (<is clc, IFBR>) {
        do_dsa:
            compute <is clc, SFDL, AILL, TILL, SILL>;
            loop (<is clc, LFOR>, mmi) {
                compute <is clc, CMLL, AILL, TILL, SILL>;
                loop (<is clc, LFOR>, jt) {
                    compute <is clc, CMLL, SILL>;
                    loop (<is clc, LFOR>, it) {
                        compute <is clc, CMLL, 6*ARD3, 3*TFDL, 4*ARD1
                            , 4*MFDL, 2*AFDL, INLL>;
                    }
                    compute <is clc, INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, 2*ARD1, AFDL, TFDL>;
        1-(do_dsa):
            compute <is clc, SFDL, SILL>;
            loop (<is clc, LFOR>, mmi) {
                compute <is clc, CMLL, AILL, TILL, SILL>;
                loop (<is clc, LFOR>, jt) {
                    compute <is clc, CMLL, SILL>;
                    loop (<is clc, LFOR>, it) {
                        compute <is clc, CMLL, 3*ARD3, 2*TFDL, 3*ARD1
                            , 3*MFDL, AFDL, INLL>;
                    }
                    compute <is clc, INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, 2*ARD1, AFDL, TFDL>;
        }
    }
    compute <is clc, 2*AILL, DILL, TILL>;
} (* End of pipeline_init *)

(* Calls: min max *)
proc cflow kk_loop_init { (* Defined at sweep.c:410 *)
    compute <is clc, FCAL, CMLL>;
    case (<is clc, IFBR>) {
    0.5:
        compute <is clc, 4*AILL, MILL, TILL>;
        call cflow min;
        compute <is clc, 2*TILL, 2*AILL>;
    1-(0.5):
        compute <is clc, 4*AILL, MILL, TILL>;
        call cflow max;
        compute <is clc, 2*TILL, 2*AILL>;
    }
    compute <is clc, 6*AILL, 4*MILL, 2*TILL>;
} (* End of kk_loop_init *)

(* Calls: sign *)
proc cflow else_ew_rcv {  (* Defined at sweep.c:471 *)
    compute <is clc, FCAL, 2*CMLL, ANDL>;
    case (<is clc, IFBR>) {
    p2:
        compute <is clc, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, SILL>;
            loop (<is clc, LFOR>, nk) {
                compute <is clc, CMLL, SILL>;
                loop (<is clc, LFOR>, jt) {
                    compute <is clc, CMLL, ARD3, SFDL, INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
    1-(p2):
        compute <is clc, SFDL, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, AILL, TILL, SILL>;
            loop (<is clc, LFOR>, nk) {
                compute <is clc, CMLL, AILL>;
                compute <is clc, AILL>;
                call cflow sign;
                compute <is clc, TILL, SILL>;
                loop (<is clc, LFOR>, jt) {
                    compute <is clc, CMLL, 3*ARD3, 2*TFDL, 3*ARD1, 3*MFDL
                        , AFDL, INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
        compute <is clc, 2*ARD1, AFDL, TFDL>;
    }
} (* End of else_ew_rcv *)

(* Calls: sign *)
proc cflow comp_face {    (* Defined at sweep.c:550 *)
    compute <is clc, FCAL>;
    case (<is clc, IFBR>) {
    do_dsa:
        compute <is clc, AILL, TILL, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, AILL, TILL, SILL>;
            loop (<is clc, LFOR>, nk) {
                compute <is clc, CMLL, AILL>;
                compute <is clc, AILL>;
                call cflow sign;
                compute <is clc, TILL, SILL>;
                loop (<is clc, LFOR>, jt) {
                    compute <is clc, CMLL, 3*ARD3, ARD1, MFDL, AFDL, TFDL
                        , INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
    }
} (* End of comp_face *)

(* Calls: sign *)
proc cflow else_ns_rcv {  (* Defined at sweep.c:620 *)
    compute <is clc, FCAL, 2*CMLL, ANDL>;
    case (<is clc, IFBR>) {
    p3:
        compute <is clc, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, SILL>;
            loop (<is clc, LFOR>, nk) {
                compute <is clc, CMLL, SILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, CMLL, ARD3, SFDL, INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
    1-(p3):
        compute <is clc, SFDL, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, AILL, TILL, SILL>;
            loop (<is clc, LFOR>, nk) {
                compute <is clc, CMLL, AILL>;
                compute <is clc, AILL>;
                call cflow sign;
                compute <is clc, TILL, SILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, CMLL, 3*ARD3, 2*TFDL, 3*ARD1, 3*MFDL
                        , AFDL, INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
        compute <is clc, 2*ARD1, AFDL, TFDL>;
    }
} (* End of else_ns_rcv *)

(* Calls: sign min max *)
proc cflow work {  (* Defined at sweep.c:697 *)
    compute <is clc, FCAL>;
    case (<is clc, IFBR>) {
    do_dsa:
        compute <is clc, AILL, TILL, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, AILL, TILL, SILL>;
            loop (<is clc, LFOR>, nk) {
                compute <is clc, CMLL, AILL>;
                compute <is clc, AILL>;
                call cflow sign;
                compute <is clc, TILL, SILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, CMLL, 3*ARD3, ARD1, MFDL, AFDL, TFDL
                        , INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
    }
    compute <is clc, SILL>;
    loop (<is clc, LFOR>, mmi) {
        compute <is clc, CMLL, ARL1, SILL, INLL>;
    }
    compute <is clc, SILL>;
    loop (<is clc, LFOR>, jt+nk-1+mmi-1) {
        compute <is clc, 4*AILL, CMLL, SILL, TILL>;
        loop (<is clc, LFOR>, mmi-1) {
            compute <is clc, CMLL, 3*ARL1, 2*TILL, AILL, INLL>;
        }
        compute <is clc, 2*AILL>;
        call cflow min;
        call cflow min;
        call cflow min;
        call cflow max;
        compute <is clc, 2*ARL1, 2*TILL, AILL, 2*SILL>;
        loop (<is clc, LFOR>, ndiag) {
            compute <is clc, CMLL, TILL, SILL>;
            loop (<is clc, LFOR>, mmi-1) {
                compute <is clc, 2*AILL, CMLL, ARL1, TILL, INLL>;
            }
            compute <is clc, 2*TILL, 3*AILL>;
            call cflow min;
            compute <is clc, AILL>;
            call cflow sign;
            compute <is clc, TILL, 3*AILL>;
            call cflow max;
            compute <is clc, AILL>;
            call cflow sign;
            compute <is clc, 3*TILL, 2*AILL, ABSI, 5*ARD1, 2*MFDL, 4*TFDL
                , ARD3, SILL>;
            loop (<is clc, LFOR>, it) {
                compute <is clc, CMLL, ARD3, ARD1, TFDL, INLL>;
            }
            compute <is clc, SILL>;
            loop (<is clc, LFOR>, nm-1) {
                compute <is clc, CMLL, SILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, CMLL, 2*ARD1, 2*ARD3, MFDL, AFDL, TFDL
                        , INLL>;
                }
                compute <is clc, INLL>;
            }
            case (<is clc, IFBR>) {
            (-ifixups)/(-epsi):
                compute <is clc, TILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, 4*CMLL, 3*ANDL, 8*ARD1, 8*MFDL, 9*TFDL
                        , 7*ARD3, 9*AFDL, DFDL, AILL, TILL>;
                }
            1-((-ifixups)/(-epsi)):
                compute <is clc, TILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, 4*CMLL, 3*ANDL, 7*ARD1, 8*MFDL, 8*TFDL
                        , 5*ARD3, 9*AFDL, DFDL, SILL, CMDL>;
                    case (<is clc, IFBR>) {
                    0.5:
                        compute <is clc, 2*AFDL, 4*TFDL, DFDL, 3*MFDL, ARD1
                            , SFDL, CMDL>;
                        case (<is clc, IFBR>) {
                        0.5:
                            compute <is clc, ARD1, MFDL, ARD3, AFDL, TFDL>;
                        }
                        compute <is clc, CMDL>;
                        case (<is clc, IFBR>) {
                        0.5:
                            compute <is clc, ARD1, MFDL, ARD3, AFDL, TFDL>;
                        }
                        compute <is clc, SILL>;
                    }
                    compute <is clc, CMDL>;
                    case (<is clc, IFBR>) {
                    0.5:
                        compute <is clc, 2*AFDL, 4*TFDL, DFDL, 3*MFDL, ARD3
                            , ARD1, SFDL, CMDL>;
                        case (<is clc, IFBR>) {
                        0.5:
                            compute <is clc, ARD1, MFDL, ARD3, AFDL, TFDL>;
                        }
                        compute <is clc, CMDL>;
                        case (<is clc, IFBR>) {
                        0.5:
                            compute <is clc, ARD1, MFDL, AFDL, TFDL>;
                        }
                        compute <is clc, SILL>;
                    }
                    compute <is clc, CMDL>;
                    case (<is clc, IFBR>) {
                    0.5:
                        compute <is clc, 2*AFDL, 4*TFDL, DFDL, 3*MFDL, ARD3
                            , ARD1, SFDL, CMDL>;
                        case (<is clc, IFBR>) {
                        0.5:
                            compute <is clc, ARD1, MFDL, AFDL, TFDL>;
                        }
                        compute <is clc, CMDL>;
                        case (<is clc, IFBR>) {
                        0.5:
                            compute <is clc, ARD1, MFDL, ARD3, AFDL, TFDL>;
                        }
                        compute <is clc, SILL>;
                    }
                    compute <is clc, 4*TFDL, ARD1, 2*ARD3, 2*AILL, 2*TILL>;
                }
            }
            compute <is clc, SILL>;
            loop (<is clc, LFOR>, it) {
                compute <is clc, CMLL, 2*ARD3, 2*ARD1, MFDL, AFDL, TFDL
                    , INLL>;
            }
            compute <is clc, SILL>;
            loop (<is clc, LFOR>, nm-1) {
                compute <is clc, CMLL, SILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, CMLL, 3*ARD3, 2*ARD1, 2*MFDL, AFDL
                        , TFDL, INLL>;
                }
                compute <is clc, INLL>;
            }
            case (<is clc, IFBR>) {
            do_dsa:
                compute <is clc, SILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, CMLL, 8*ARD3, 4*ARD1, 3*MFDL, 3*AFDL
                        , 3*TFDL, INLL>;
                }
            }
```

```
            compute <is clc, ARD3, TFDL, INLL>;
            compute <is clc, 2*POL1, AILL, TILL, INLL>;
        }
    }
} (* End of work *)

(* Calls: sign *)
proc cflow else_ew_snd {  (* Defined at sweep.c:996 *)
    compute <is clc, FCAL, 2*CMLL, ANDL>;
    case (<is clc, IFBR>) {
    1-p2:
        compute <is clc, SFDL, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, AILL, TILL, SILL>;
            loop (<is clc, LFOR>, nk) {
                compute <is clc, CMLL, AILL>;
                compute <is clc, AILL>;
                call cflow sign;
                compute <is clc, TILL, SILL>;
                loop (<is clc, LFOR>, jt) {
                    compute <is clc, CMLL, 3*ARD3, 2*TFDL, 3*ARD1, 3*MFDL
                        , AFDL, INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
        compute <is clc, 2*ARD1, AFDL, TFDL>;
    1-(1-p2):
        compute <is clc, SFDL, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, AILL, TILL, SILL>;
            loop (<is clc, LFOR>, nk) {
                compute <is clc, CMLL, AILL>;
                compute <is clc, AILL>;
                call cflow sign;
                compute <is clc, TILL, SILL>;
                loop (<is clc, LFOR>, jt) {
                    compute <is clc, CMLL, 3*ARD1, ARD3, 3*MFDL, AFDL, TFDL
                        , INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
        compute <is clc, 2*ARD1, AFDL, TFDL>;
    }
} (* End of else_ew_snd *)

(* Calls: sign *)
proc cflow else_ns_snd {  (* Defined at sweep.c:1082 *)
    compute <is clc, FCAL, 2*CMLL, ANDL>;
    case (<is clc, IFBR>) {
    1-p3:
        compute <is clc, SFDL, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, AILL, TILL, SILL>;
            loop (<is clc, LFOR>, nk) {
                compute <is clc, CMLL, AILL>;
                compute <is clc, AILL>;
                call cflow sign;
                compute <is clc, TILL, SILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, CMLL, 3*ARD3, 2*TFDL, 3*ARD1, 3*MFDL
                        , AFDL, INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
        compute <is clc, 2*ARD1, AFDL, TFDL>;
    1-(1-p3):
        compute <is clc, SFDL, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, AILL, TILL, SILL>;
            loop (<is clc, LFOR>, nk) {
                compute <is clc, CMLL, AILL>;
                compute <is clc, AILL>;
                call cflow sign;
                compute <is clc, TILL, SILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, CMLL, 3*ARD1, ARD3, 3*MFDL, AFDL, TFDL
                        , INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
        compute <is clc, 2*ARD1, AFDL, TFDL>;
    }
} (* End of else_ns_snd *)

(* Calls: *)
proc cflow last {  (* Defined at sweep.c:1178 *)
    compute <is clc, FCAL, 2*CMLL, ANDL>;
    case (<is clc, IFBR>) {
    1-p1:
        compute <is clc, SFDL, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, AILL, TILL, SILL>;
            loop (<is clc, LFOR>, jt) {
                compute <is clc, CMLL, SILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, CMLL, 3*ARD3, 2*TFDL, 3*ARD1, 3*MFDL
                        , AFDL, INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
        compute <is clc, 2*ARD1, AFDL, TFDL>;
    1-(1-p1):
        compute <is clc, SFDL, SILL>;
        loop (<is clc, LFOR>, mmi) {
            compute <is clc, CMLL, AILL, TILL, SILL>;
            loop (<is clc, LFOR>, jt) {
                compute <is clc, CMLL, SILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, CMLL, 3*ARD1, ARD3, 3*MFDL, AFDL, TFDL
                        , INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
        compute <is clc, 2*ARD1, AFDL, TFDL>;
    }
} (* End of last *)

(* Calls: *)
proc cflow max {   (* Defined at sweep.c:1263 *)
    compute <is clc, FCAL, 2*FARD, CMDL>;
    case (<is clc, IFBR>) {
    0.5:
        compute <0>;
    1-(0.5):
        compute <0>;
    }
    return;
} (* End of max *)

(* Calls: *)
proc cflow min {   (* Defined at sweep.c:1264 *)
    compute <is clc, FCAL, 2*FARD, CMDL>;
    case (<is clc, IFBR>) {
    0.5:
        compute <0>;
    1-(0.5):
        compute <0>;
    }
    return;
} (* End of min *)

(* Calls: *)
```

```
    proc cflow sign {  (* Defined at sweep.c:1265 *)
        compute <is clc, FCAL, 2*FARD, CMDL>;
        case (<is clc, IFBR>) {
        0.5:
            compute <is clc, ABSD>;
        1-(0.5):
            compute <is clc, ABSD>;
        }
        return;
    } (* End of sign *)


}
```

A.4 Subtask Object: fixed

```
subtask fixed {
        include hardware;
        include globalsum;

        link {
        globalsum:
                Tx_sum = sum_fixed (),
                Tx_comp = comp_fixup ();
        }

(*
 * CHIP3S
 * Application Characterisation Tool
 * Source  : fixed.c
 * RUV Type: clc
 *)

    (* Calls: *)
    proc cflow sum_fixed {    (* Defined at fixed.c:1 *)
        compute <is clc, FCAL, 2*POL1, AILL, TILL>;
    } (* End of sum_fixed *)

    (* Calls: *)
    proc cflow comp_fixup {   (* Defined at fixed.c:8 *)
        compute <is clc, FCAL, AILL, TILL>;
    } (* End of comp_fixup *)
}
```

A.5 Subtask Object: flux_err

```
subtask flux_err {
        include hardware;
        include globalmax;

        var numeric:
                it = 25,
                jt = 17,
                kt = 50;

        link {
        globalmax:
                Tx_comp = comp_flux_err(),
                Tx_max = max_flux_err();
        }

(*
 * CHIP3S
 * Application Characterisation Tool
 * Source  : flux_err.c
 * RUV Type: clc
 *)

    (* Calls: max *)
    proc cflow comp_flux_err {       (* Defined at flux_err.c:1 *)
```

```
        compute <is clc, FCAL, POD1, SFDL, SILL>;
        loop (<is clc, LFOR>, kt) {
            compute <is clc, CMLL, SILL>;
            loop (<is clc, LFOR>, jt) {
                compute <is clc, CMLL, SILL>;
                loop (<is clc, LFOR>, it) {
                    compute <is clc, CMLL, ARD3, CMDL>;
                    case (<is clc, IFBR>) {
                    0.5:
                        compute <is clc, 3*ARD3, AFDL, DFDL, ABSD, TFDL, POD1>;
                        call cflow max;
                        compute <is clc, POD1, TFDL>;
                    }
                    compute <is clc, INLL>;
                }
                compute <is clc, INLL>;
            }
            compute <is clc, INLL>;
        }
    } (* End of comp_flux_err *)

    (* Calls: max *)
    proc cflow max_flux_err { (* Defined at flux_err.c:37 *)
        compute <is clc, FCAL, POD1>;
        call cflow max;
        compute <is clc, POD1, TFDL>;
    } (* End of max_flux_err *)

    (* Calls: *)
    proc cflow max {   (* Defined at flux_err.c:43 *)
        compute <is clc, FCAL, 2*FARD, CMDL>;
        case (<is clc, IFBR>) {
        0.5:
            compute <0>;
        1-(0.5):
            compute <0>;
        }
        return;
    } (* End of max *)

}
```

A.6 Parallel Template Object: async

```
(*
 * async.la - Sequential 'parallel' template
 *)

partmp async {

        include hardware;

        var compute: Tx;

        option {
                nstage = 1,
                seval  = 0
        }

        proc exec init {
                step cpu {
                        confdev Tx;
                }
        }
}
```

A.7 Parallel Template Object: pipeline

```
#include <mpidefs.h>
partmp pipeline {
```

```
include hardware;
include Eval;

var compute:
       Tx_sweep_init,
       Tx_octant,
       Tx_get_direct,
       Tx_pipeline_init,
       Tx_kk_loop_init,
       Tx_else_ew_rcv,
       Tx_comp_face,
       Tx_else_ns_rcv,
       Tx_work,
       Tx_else_ew_snd,
       Tx_else_ns_snd,
       Tx_last;

var numeric:
       mm = 6,
       mmi = 3,
       it = 26,
       jt = 18,
       kt = 50,
       mk = 10,
       npe_i = 2,
       npe_j = 3;

option {
       nstage = 1,
       seval  = 0;
}

proc exec Get_i2
       var phase;
{
       var numeric:
              i2;
       if ( phase <= 4) i2 = -1;
       else i2 = 1;
       return i2;
}

proc exec Get_j2
       var phase;
{
       var numeric:
              j2;
       if ( phase == 1) j2 = -1;
       else if(phase == 2) j2 = -1;
       else if(phase == 3) j2 = 1;
       else if(phase == 4) j2 = 1;
       else if(phase == 5) j2 = -1;
       else if(phase == 6) j2 = -1;
       else if(phase == 7) j2 = 1;
       else j2 = 1;
       return j2;
}

proc exec Get_myid
       var x, y;
{
       var numeric:
              myid;
       myid = npe_i * (y - 1) + x;
       return myid;
}

proc exec West
       var x, y;
{
       var numeric:
              west;
       west  = 0;
       if ( x != 1 ) west  = Get_myid( x-1, y );
       return west;
}
```

```
proc exec East
        var x, y;
{
        var numeric:
                east;
        east  = 0;
        if ( x != npe_i ) east  = Get_myid( x+1, y );
        return east;
}

proc exec South
        var x, y;
{
        var numeric:
                south;
        south = 0;
        if ( y != 1 ) south = Get_myid( x, y-1);
        return south;
}

proc exec North
        var x, y;
{
        var numeric:
                north;
        north = 0;
        if ( y != npe_j ) north = Get_myid( x, y+1 );
        return north;
}

proc exec Get_ew_rcv
        var phase, x, y;
{
        var numeric:
                i2, ew_rcv;

        i2 =  Get_i2( phase );
        if (i2 > 0)
                ew_rcv = West( x, y );
        else
                ew_rcv = East( x, y );
        return ew_rcv;
}

proc exec Get_ns_rcv
        var phase, x, y;
{
        var numeric:
                j2, ns_rcv;

        j2 =  Get_j2( phase );
        if (j2 > 0)
                ns_rcv = South( x, y );
        else
                ns_rcv = North( x, y );
        return ns_rcv;
}

proc exec Get_ew_snd
        var phase, x, y;
{
        var numeric:
                i2, ew_snd;

        i2 =  Get_i2( phase );
        if (i2 > 0)
                ew_snd = East( x, y );
        else
                ew_snd = West( x, y );
        return ew_snd;
}

proc exec Get_ns_snd
        var phase, x, y;
{
        var numeric:
                j2, ns_snd;

        j2 =  Get_j2( phase );
        if (j2 > 0)
                ns_snd = North( x, y );
        else
                ns_snd = South( x, y );
        return ns_snd;
}


proc exec init {
        var numeric:
                phase,
                i,
                j,
                x,
                y,
                myid,
                mmo,
                kb,
                nib,
                njb,
                ew_rcv,
                ns_rcv,
                ew_snd,
                ns_snd;

        mmo = mm/mmi;
        kb = ( kt + mk - 1 ) / mk;
        nib = (jt+1)*(mk+1)*(mmi+1);
        njb = (it+1)*(mk+1)*(mmi+1);

        step cpu {
                confdev Tx_sweep_init;
        }

        for( phase = 1; phase <= 8; phase = phase + 1)
        {
                step cpu {
                        confdev Tx_octant;
                }

                step cpu {
                        confdev Tx_get_direct;
                }

                for( i = 1; i <= mmo; i = i + 1 )
                {
                        step cpu {
                                confdev Tx_pipeline_init;
                        }

                        for( j = 1; j <= kb; j = j + 1 )
                        {
                                step cpu {
                                        confdev Tx_kk_loop_init;
                                }

                                for( x = 1; x <= npe_i; x = x + 1 )
                                for( y = 1; y <= npe_j; y = y + 1 )
                                {
                                        myid = Get_myid( x, y );
                                        ew_rcv = Get_ew_rcv( phase, x, y );
                                        if( ew_rcv != 0 )
                                        {
                                                step mpirecv {
                                                        confdev  ew_rcv,  myid,
                                                        nib, MPI_Packed;
                                                }
                                        }
                                        else
                                        {
                                                step cpu on myid {
                                                       confdev  Tx_else_ew_rcv;
                                                }
                                        }
                                }

                                step cpu {
                                        confdev Tx_comp_face;
                                }

                                for( x = 1; x <= npe_i; x = x + 1 )
                                for( y = 1; y <= npe_j; y = y + 1 )
                                {
                                        myid = Get_myid( x, y );
                                        ns_rcv = Get_ns_rcv( phase, x, y );
                                        if( ns_rcv != 0 )
                                        {
                                                step mpirecv {
                                                        confdev  ns_rcv,  myid,
                                                        njb, MPI_Packed;
                                                }
                                        }
                                        else
                                        {
                                                step cpu on myid {
                                                       confdev  Tx_else_ns_rcv;
                                                }
                                        }

                                }

                                step cpu {
                                        confdev Tx_work;
                                }

                                for( x = 1; x <= npe_i; x = x + 1 )
                                for( y = 1; y <= npe_j; y = y + 1 )
                                {
                                        myid = Get_myid( x, y );
                                        ew_snd = Get_ew_snd( phase, x, y );
                                        if( ew_snd != 0 )
                                        {
                                                step mpisend {
                                                        confdev  myid,  ew_snd,
                                                        nib, MPI_Packed;
                                                }
                                        }
                                        else
                                        {
                                                step cpu on myid {
                                                       confdev  Tx_else_ew_snd;
                                                }
                                        }

                                }

                                for( x = 1; x <= npe_i; x = x + 1 )
                                for( y = 1; y <= npe_j; y = y + 1 )
                                {
                                        myid = Get_myid( x, y );
                                        ns_snd = Get_ns_snd( phase, x, y );
                                        if( ns_snd != 0 )
                                        {
                                                step mpisend {
                                                        confdev  myid,  ns_snd,
                                                        njb, MPI_Packed;
                                                }
                                        }
                                        else
                                        {
                                                step cpu on myid {
                                                       confdev  Tx_else_ns_snd;
                                                }
                                        }

                                }

                        }

                        step cpu {
```

```
                                        confdev Tx_last;
                                }
                        }
                }
        }
}
```

A.8 Parallel Template Object: globalsum

```
#include <mpidefs.h>
partmp globalsum {

        include hardware;

        var compute:
                Tx_sum,
                Tx_comp;

        option {
                nstage = 1,
                seval  = 0
        }

        proc exec init {
                var numeric: i,j;


                for( i = 2; i <= hardware.Nproc; i = i + 1)
                {
                        step mpisend {
                                confdev i, 1, 1, MPI_Packed;
                        }

                        step mpirecv {
                                confdev i, 1, 1, MPI_Packed;
                        }
                        step cpu on 1 {
                                confdev Tx_sum;
                        }
                }


                for( i = 2; i <= hardware.Nproc; i = i + 1)
                {
                        step mpisend {
                                confdev 1,i,1, MPI_Packed;
                        }


                        step mpirecv {
                                confdev 1,i,1, MPI_Packed;
                        }

                }

                step cpu {
                        confdev Tx_comp;
                }
        }
}
```

A.9 Parallel Template Object: globalmax

```
#include <mpidefs.h>
partmp globalmax {

        include hardware;

        var compute:
                Tx_max,
                Tx_comp;

        option {
                nstage = 1,
                seval  = 0;
        }

        proc exec init {
                var numeric: i;


                step cpu {
                        confdev Tx_comp;
                }

                for( i = 2; i <= hardware.Nproc; i = i + 1)
                {
                        step mpisend {
                                confdev i, 1, 1, MPI_Packed;
                        }

                        step mpirecv {
                                confdev i, 1, 1, MPI_Packed;
                        }
                        step cpu on 1 {
                                confdev Tx_max;
                        }
                }


                for( i = 2; i <= hardware.Nproc; i = i + 1)
                {
                        step mpisend {
                                confdev 1,i,1, MPI_Packed;
                        }


                        step mpirecv {
                                confdev 1,i,1, MPI_Packed;
                        }

                }
        }
}
```

Appendix B

ARMS EXPERIMENT RESULTS

The ARMS experiment results are included in this section. There are totally 149 application execution requests sent to the agent system, 144 of them are executed and 5 of them are failed for resource discovery.

The user view of the results is shown in Section B.1, which includes all the applications requests (including application ID, application name, and required time) and their execution details (including discovery agents, discovery time, waiting time, execution time, and the number of processors used) during the experiment.

There are 8 agents in the experimental system. The agent views of the results are shown in Section B.2 – B.9 respectively. In each agent view, there is an application browser and a correspondent Gantt chart. Note that each agent identifies an incoming application using a new unique ID, which may be not same as those shown in the user view. And also note that the Gantt chart only gives a graphical view of up to latest 16 applications that are scheduled on an agent.

B.1 Experiment Results @ Users

ID	Application Name	RT	Discovery Agents	DT	WT	ET	#P
52420	/dcs/vlsi/junwei/a4/arms/memsort	30	origin-->found	0	0	10	8
52422	/dcs/vlsi/junwei/a4/arms/cpi	3	tizer-->gem-->found	5	0	2	12
52425	/dcs/vlsi/junwei/a4/arms/improc	138	sprite-->found	0	0	40	8
52426	/dcs/vlsi/junwei/a4/arms/fft	24	rubbish-->tizer-->found	3	0	20	16
52430	/dcs/vlsi/junwei/a4/arms/jacobi	47	coke-->found	0	0	19	15
52432	/dcs/vlsi/junwei/a4/arms/memsort	27	tizer-->gem-->found	4	0	10	8
52433	/dcs/vlsi/junwei/a4/arms/cpi	120	origin-->found	0	0	2	12
52436	/dcs/vlsi/junwei/a4/arms/sweep3d	94	burroughs-->found	0	0	14	15
52439	/dcs/vlsi/junwei/a4/arms/fft	98	burroughs-->found	0	11	36	16
52441	/dcs/vlsi/junwei/a4/arms/jacobi	38	coke-->found	0	8	19	15
52445	/dcs/vlsi/junwei/a4/arms/fft	40	tizer-->found	0	4	20	16
52448	/dcs/vlsi/junwei/a4/arms/memsort	45	budweiser-->found	0	0	24	8
52451	/dcs/vlsi/junwei/a4/arms/jacobi	56	origin-->found	0	0	6	15
52455	/dcs/vlsi/junwei/a4/arms/cpi	2	tizer-->gem-->failed	11	-	-	-
52458	/dcs/vlsi/junwei/a4/arms/sweep3d	120	tizer-->found	0	11	8	15
52459	/dcs/vlsi/junwei/a4/arms/sweep3d	55	rubbish-->found	0	0	16	15
52461	/dcs/vlsi/junwei/a4/arms/closure	26	rubbish-->found	0	14	8	15
52463	/dcs/vlsi/junwei/a4/arms/jacobi	76	burroughs-->found	0	23	21	15
52467	/dcs/vlsi/junwei/a4/arms/improc	59	burroughs-->tizer-->found	6	16	40	8
52469	/dcs/vlsi/junwei/a4/arms/jacobi	116	tizer-->found	0	8	12	15
52473	/dcs/vlsi/junwei/a4/arms/improc	160	burroughs-->found	0	34	72	8
52477	/dcs/vlsi/junwei/a4/arms/cpi	55	sprite-->found	0	0	4	12
52480	/dcs/vlsi/junwei/a4/arms/cpi	113	gem-->found	0	0	2	12
52484	/dcs/vlsi/junwei/a4/arms/jacobi	123	origin-->found	0	0	6	15
52486	/dcs/vlsi/junwei/a4/arms/closure	9	sprite-->found	0	0	4	15
52490	/dcs/vlsi/junwei/a4/arms/sweep3d	172	coke-->found	0	0	12	15
52491	/dcs/vlsi/junwei/a4/arms/fft	38	burroughs-->tizer-->found	6	0	36	8
52496	/dcs/vlsi/junwei/a4/arms/memsort	38	rubbish-->tizer-->gem-->found	10	0	10	8
52497	/dcs/vlsi/junwei/a4/arms/improc	75	coke-->found	0	5	64	8
52499	/dcs/vlsi/junwei/a4/arms/sweep3d	12	sprite-->found	0	0	8	15
52500	/dcs/vlsi/junwei/a4/arms/jacobi	40	origin-->found	0	0	6	15
52503	/dcs/vlsi/junwei/a4/arms/improc	102	gem-->found	0	0	20	8
52505	/dcs/vlsi/junwei/a4/arms/cpi	46	sprite-->found	0	2	4	12
52506	/dcs/vlsi/junwei/a4/arms/jacobi	14	sprite-->gem-->sprite-->found	13	0	12	15
52510	/dcs/vlsi/junwei/a4/arms/closure	7	gem-->sprite-->found	4	0	4	15
52514	/dcs/vlsi/junwei/a4/arms/closure	31	rubbish-->found	0	0	8	15
52517	/dcs/vlsi/junwei/a4/arms/closure	15	rubbish-->found	0	5	8	15

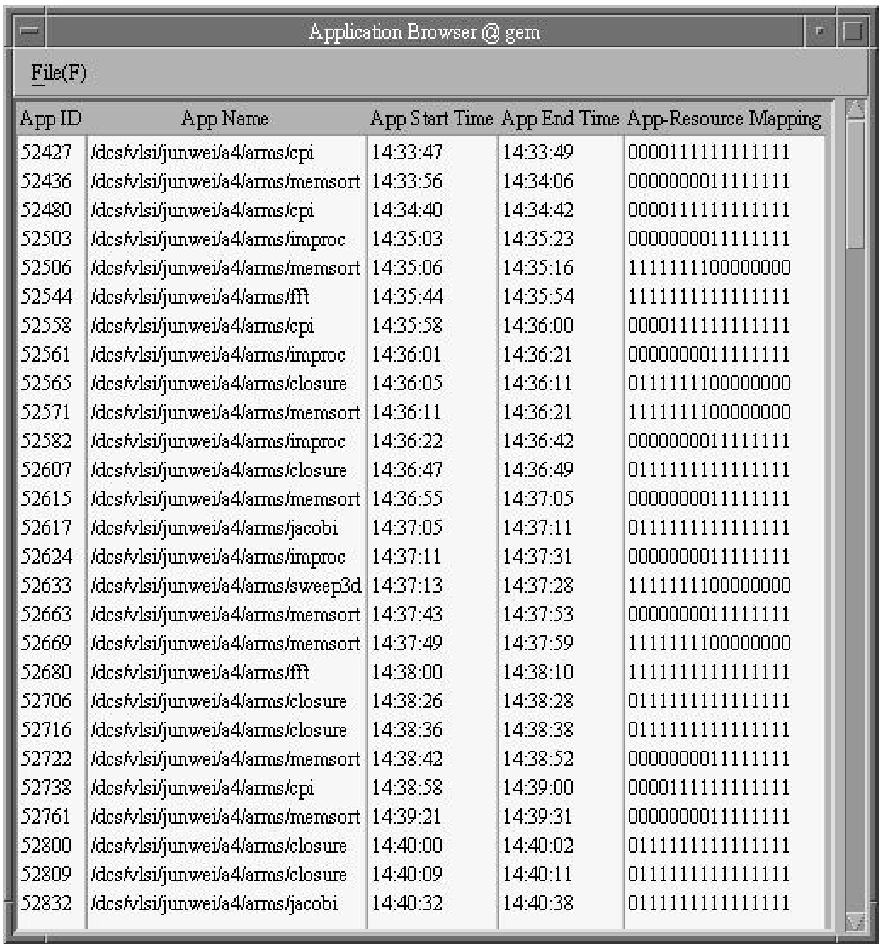
52521	/dcs/vlsi/junwei/a4/arms/memsort	52	budweiser-->found	0	0	24	8
52525	/dcs/vlsi/junwei/a4/arms/jacobi	48	budweiser-->found	0	20	14	15
52529	/dcs/vlsi/junwei/a4/arms/sweep3d	77	budweiser-->found	0	30	9	15
52532	/dcs/vlsi/junwei/a4/arms/improc	130	sprite-->found	0	0	40	8
52533	/dcs/vlsi/junwei/a4/arms/sweep3d	149	burroughs-->found	0	0	54	8
52535	/dcs/vlsi/junwei/a4/arms/improc	74	rubbish-->tizer-->found	2	0	40	8
52536	/dcs/vlsi/junwei/a4/arms/fft	30	budweiser-->sprite-->gem -->found	8	0	10	16
52538	/dcs/vlsi/junwei/a4/arms/sweep3d	56	rubbish-->found	0	0	16	15
52542	/dcs/vlsi/junwei/a4/arms/cpi	39	tizer-->found	0	0	14	8
52545	/dcs/vlsi/junwei/a4/arms/cpi	83	burroughs-->found	0	42	7	12
52546	/dcs/vlsi/junwei/a4/arms/fft	65	rubbish-->found	0	8	40	16
52550	/dcs/vlsi/junwei/a4/arms/improc	150	burroughs-->found	0	44	72	8
52551	/dcs/vlsi/junwei/a4/arms/closure	31	burroughs-->tizer-->found	9	0	12	7
52554	/dcs/vlsi/junwei/a4/arms/improc	173	budweiser-->found	0	14	48	8
52558	/dcs/vlsi/junwei/a4/arms/cpi	8	gem-->found	0	0	2	12
52561	/dcs/vlsi/junwei/a4/arms/improc	110	gem-->found	0	0	20	8
52563	/dcs/vlsi/junwei/a4/arms/jacobi	131	coke-->found	0	3	19	15
52565	/dcs/vlsi/junwei/a4/arms/closure	11	gem-->found	0	0	6	7
52567	/dcs/vlsi/junwei/a4/arms/sweep3d	166	sprite-->found	0	5	8	15
52571	/dcs/vlsi/junwei/a4/arms/memsort	16	gem-->found	0	0	10	8
52572	/dcs/vlsi/junwei/a4/arms/memsort	57	rubbish-->tizer-->found	2	0	20	8
52576	/dcs/vlsi/junwei/a4/arms/fft	40	tizer-->found	0	1	36	8
52578	/dcs/vlsi/junwei/a4/arms/fft	46	coke-->found	0	7	32	16
52582	/dcs/vlsi/junwei/a4/arms/improc	87	gem-->found	0	0	20	8
52586	/dcs/vlsi/junwei/a4/arms/memsort	37	tizer-->found	0	8	20	8
52589	/dcs/vlsi/junwei/a4/arms/fft	43	rubbish-->tizer-->found	5	20	20	16
52593	/dcs/vlsi/junwei/a4/arms/cpi	55	origin-->found	0	0	2	12
52596	/dcs/vlsi/junwei/a4/arms/improc	94	rubbish-->found	0	0	80	8
52598	/dcs/vlsi/junwei/a4/arms/improc	98	sprite-->found	0	0	40	8
52599	/dcs/vlsi/junwei/a4/arms/sweep3d	112	budweiser-->found	0	17	9	15
52602	/dcs/vlsi/junwei/a4/arms/closure	4	tizer-->gem-->found	5	0	2	15
52606	/dcs/vlsi/junwei/a4/arms/memsort	18	budweiser-->sprite-->gem-->found	9	0	10	8
52609	/dcs/vlsi/junwei/a4/arms/memsort	45	budweiser-->found	0	16	24	8
52613	/dcs/vlsi/junwei/a4/arms/closure	2	origin-->gem-->failed	8	-	-	-
52617	/dcs/vlsi/junwei/a4/arms/jacobi	68	gem-->found	0	8	6	15
52621	/dcs/vlsi/junwei/a4/arms/improc	167	tizer-->found	0	13	40	8
52624	/dcs/vlsi/junwei/a4/arms/improc	89	gem-->found	0	7	20	8
52628	/dcs/vlsi/junwei/a4/arms/memsort	35	budweiser-->found	0	0	24	8
52633	/dcs/vlsi/junwei/a4/arms/sweep3d	110	gem-->found	0	0	15	8
52637	/dcs/vlsi/junwei/a4/arms/improc	91	rubbish-->found	0	0	80	8
52641	/dcs/vlsi/junwei/a4/arms/sweep3d	117	tizer-->found	0	0	30	8
52645	/dcs/vlsi/junwei/a4/arms/memsort	53	sprite-->found	0	0	20	8
52648	/dcs/vlsi/junwei/a4/arms/improc	91	origin-->found	0	0	20	8
52649	/dcs/vlsi/junwei/a4/arms/memsort	23	coke-->sprite-->found	5	0	20	8

52652	/dcs/vlsi/junwei/a4/arms/memsort	17	burroughs-->tizer-->gem-->found	17	0	10	8
52654	/dcs/vlsi/junwei/a4/arms/memsort	26	rubbish-->tizer-->gem-->found	9	0	10	8
52658	/dcs/vlsi/junwei/a4/arms/sweep3d	121	origin-->found	0	0	4	15
52660	/dcs/vlsi/junwei/a4/arms/sweep3d	77	burroughs-->found	0	6	14	15
52661	/dcs/vlsi/junwei/a4/arms/fft	80	coke-->found	0	0	32	16
52664	/dcs/vlsi/junwei/a4/arms/fft	86	origin-->found	0	8	10	16
52666	/dcs/vlsi/junwei/a4/arms/closure	13	burroughs-->tizer-->found	6	2	4	15
52671	/dcs/vlsi/junwei/a4/arms/improc	91	sprite-->found	0	0	40	8
52673	/dcs/vlsi/junwei/a4/arms/closure	25	rubbish-->tizer-->found	2	3	4	15
52677	/dcs/vlsi/junwei/a4/arms/closure	3	sprite-->gem-->failed	12	-	-	-
52678	/dcs/vlsi/junwei/a4/arms/closure	8	sprite-->budweiser-->found	16	0	4	15
52680	/dcs/vlsi/junwei/a4/arms/fft	91	gem-->found	0	0	10	16
52684	/dcs/vlsi/junwei/a4/arms/improc	190	tizer-->found	0	0	40	8
52688	/dcs/vlsi/junwei/a4/arms/fft	30	origin-->found	0	0	10	16
52691	/dcs/vlsi/junwei/a4/arms/jacobi	37	coke-->found	0	2	19	15
52694	/dcs/vlsi/junwei/a4/arms/sweep3d	58	coke-->found	0	18	12	15
52695	/dcs/vlsi/junwei/a4/arms/cpi	10	coke-->budweiser-->found	13	0	4	12
52696	/dcs/vlsi/junwei/a4/arms/improc	174	tizer-->found	0	0	40	8
52697	/dcs/vlsi/junwei/a4/arms/closure	4	rubbish-->tizer-->gem-->found	9	0	2	15
52702	/dcs/vlsi/junwei/a4/arms/improc	140	origin-->found	0	0	20	8
52703	/dcs/vlsi/junwei/a4/arms/fft	69	sprite--> found	0	8	20	16
52704	/dcs/vlsi/junwei/a4/arms/memsort	10	tizer-->gem-->failed	15	-	-	-
52709	/dcs/vlsi/junwei/a4/arms/jacobi	46	origin-->found	0	0	15	8
52711	/dcs/vlsi/junwei/a4/arms/memsort	19	budweiser-->sprite-->gem-->found	11	0	10	8
52713	/dcs/vlsi/junwei/a4/arms/improc	167	sprite--> found	0	18	40	8
52715	/dcs/vlsi/junwei/a4/arms/memsort	57	budweiser-->found	0	0	24	8
52716	/dcs/vlsi/junwei/a4/arms/closure	27	gem-->found	0	0	2	15
52718	/dcs/vlsi/junwei/a4/arms/sweep3d	153	origin-->found	0	6	4	15
52724	/dcs/vlsi/junwei/a4/arms/closure	11	coke-->found	0	0	6	15
52729	/dcs/vlsi/junwei/a4/arms/closure	25	origin-->found	0	0	2	15
52731	/dcs/vlsi/junwei/a4/arms/cpi	2	burroughs-->tizer-->gem-->failed	24	-	-	-
52732	/dcs/vlsi/junwei/a4/arms/cpi	47	tizer-->found	0	4	4	12
52737	/dcs/vlsi/junwei/a4/arms/improc	191	burroughs-->found	0	0	72	8
52738	/dcs/vlsi/junwei/a4/arms/cpi	88	gem-->found	0	0	2	12
52740	/dcs/vlsi/junwei/a4/arms/cpi	83	budweiser-->found	0	0	4	12
52745	/dcs/vlsi/junwei/a4/arms/cpi	31	coke-->found	0	0	6	12
52747	/dcs/vlsi/junwei/a4/arms/memsort	21	coke-->sprite-->gem-->found	14	0	10	8
52749	/dcs/vlsi/junwei/a4/arms/closure	5	budweiser-->found	0	0	4	15

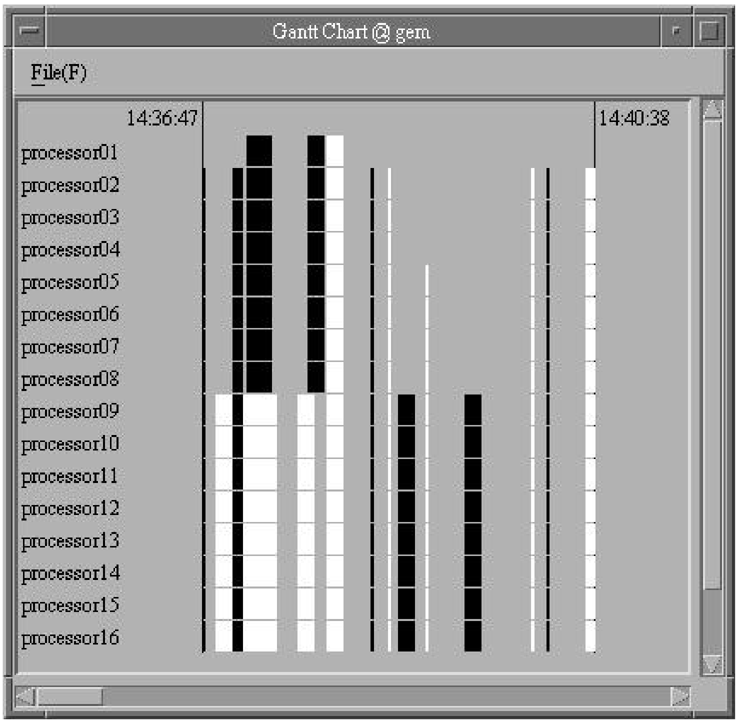
52751	/dcs/vlsi/junwei/a4/arms/cpi	98	sprite--> found	0	0	14	8
52754	/dcs/vlsi/junwei/a4/arms/sweep3d	79	coke-->found	0	0	12	15
52756	/dcs/vlsi/junwei/a4/arms/closure	25	budweiser-->found	0	0	4	15
52759	/dcs/vlsi/junwei/a4/arms/sweep3d	62	tizer-->found	0	0	8	15
52763	/dcs/vlsi/junwei/a4/arms/closure	34	rubbish-->found	0	0	8	15
52765	/dcs/vlsi/junwei/a4/arms/jacobi	34	coke-->found	0	1	19	15
52769	/dcs/vlsi/junwei/a4/arms/fft	58	coke-->found	0	16	32	16
52770	/dcs/vlsi/junwei/a4/arms/improc	45	sprite--> found	0	0	40	8
52774	/dcs/vlsi/junwei/a4/arms/cpi	80	sprite--> found	0	0	14	8
52775	/dcs/vlsi/junwei/a4/arms/memsort	35	origin-->found	0	0	10	8
52777	/dcs/vlsi/junwei/a4/arms/jacobi	31	rubbish-->found	0	0	24	15
52781	/dcs/vlsi/junwei/a4/arms/fft	79	budweiser-->found	0	0	24	16
52782	/dcs/vlsi/junwei/a4/arms/jacobi	153	tizer-->found	0	0	12	15
52787	/dcs/vlsi/junwei/a4/arms/memsort	67	burroughs-->found	0	0	36	8
52790	/dcs/vlsi/junwei/a4/arms/sweep3d	139	budweiser-->found	0	15	9	15
52792	/dcs/vlsi/junwei/a4/arms/closure	15	tizer-->found	0	2	4	15
52795	/dcs/vlsi/junwei/a4/arms/memsort	42	burroughs-->tizer-->found	5	0	20	8
52798	/dcs/vlsi/junwei/a4/arms/memsort	50	sprite--> found	0	0	20	8
52800	/dcs/vlsi/junwei/a4/arms/closure	30	gem-->found	0	0	2	15
52802	/dcs/vlsi/junwei/a4/arms/closure	3	tizer-->gem-->found	7	0	2	15
52807	/dcs/vlsi/junwei/a4/arms/cpi	93	sprite--> found	0	11	4	12
52809	/dcs/vlsi/junwei/a4/arms/closure	13	burroughs-->tizer-->found	5	6	4	15
52811	/dcs/vlsi/junwei/a4/arms/closure	13	budweiser-->found	0	3	4	15
52814	/dcs/vlsi/junwei/a4/arms/sweep3d	152	rubbish-->found	0	0	16	15
52816	/dcs/vlsi/junwei/a4/arms/jacobi	7	coke-->sprite-->gem-->found	16	0	6	15
52820	/dcs/vlsi/junwei/a4/arms/memsort	45	tizer-->found	0	4	20	8
52822	/dcs/vlsi/junwei/a4/arms/closure	9	burroughs-->found	0	1	7	15
52825	/dcs/vlsi/junwei/a4/arms/sweep3d	153	burroughs-->found	0	5	14	15
52828	/dcs/vlsi/junwei/a4/arms/improc	55	rubbish-->tizer-->found	3	0	40	8
52831	/dcs/vlsi/junwei/a4/arms/cpi	121	coke-->found	0	0	6	12

RT: Required Time
DT: Discovery Time
WT: Waiting Time
ET: Execution Time
#P: The Number of Processors Used

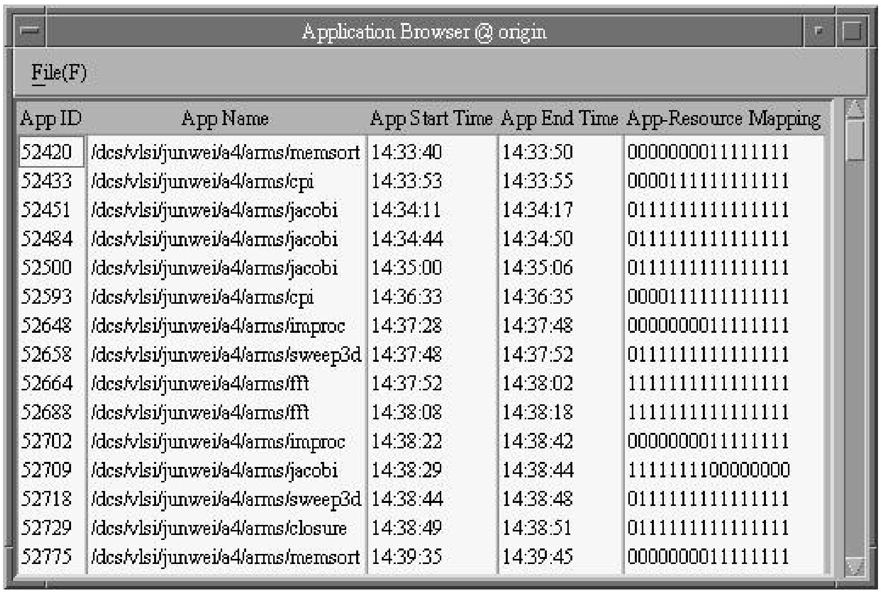
B.2 Experiment Results @ gem



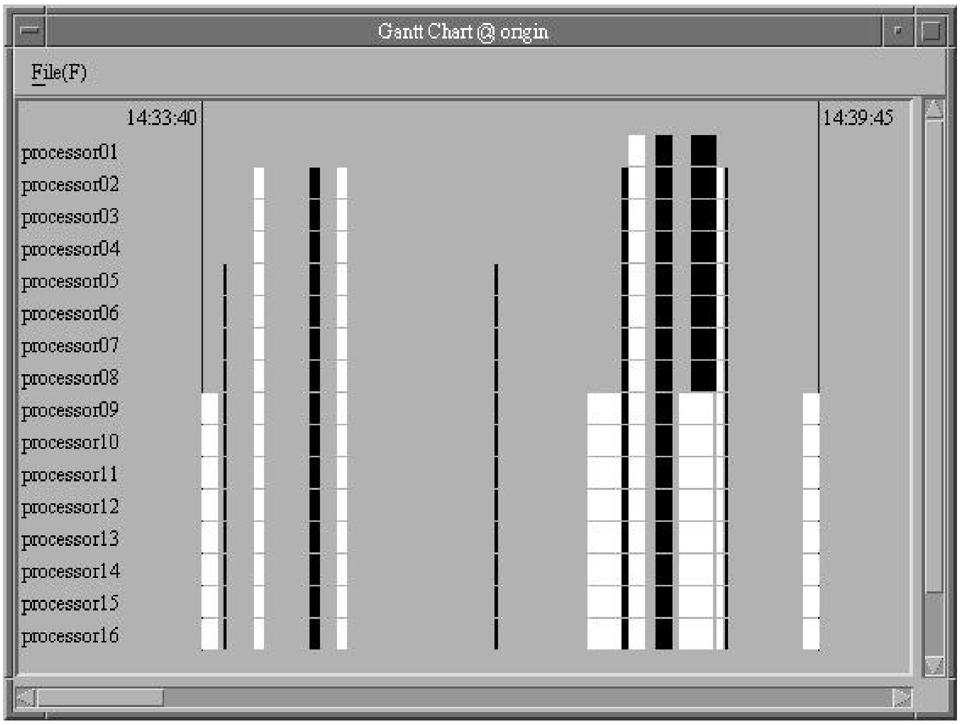
App ID	App Name	App Start Time	App End Time	App-Resource Mapping
52427	/dcs/vlsi/junwei/a4/arms/cpi	14:33:47	14:33:49	0000111111111111
52436	/dcs/vlsi/junwei/a4/arms/memsort	14:33:56	14:34:06	0000000011111111
52480	/dcs/vlsi/junwei/a4/arms/cpi	14:34:40	14:34:42	0000111111111111
52503	/dcs/vlsi/junwei/a4/arms/improc	14:35:03	14:35:23	0000000011111111
52506	/dcs/vlsi/junwei/a4/arms/memsort	14:35:06	14:35:16	1111111100000000
52544	/dcs/vlsi/junwei/a4/arms/fft	14:35:44	14:35:54	1111111111111111
52558	/dcs/vlsi/junwei/a4/arms/cpi	14:35:58	14:36:00	0000111111111111
52561	/dcs/vlsi/junwei/a4/arms/improc	14:36:01	14:36:21	0000000011111111
52565	/dcs/vlsi/junwei/a4/arms/closure	14:36:05	14:36:11	0111111100000000
52571	/dcs/vlsi/junwei/a4/arms/memsort	14:36:11	14:36:21	1111111100000000
52582	/dcs/vlsi/junwei/a4/arms/improc	14:36:22	14:36:42	0000000011111111
52607	/dcs/vlsi/junwei/a4/arms/closure	14:36:47	14:36:49	0111111111111111
52615	/dcs/vlsi/junwei/a4/arms/memsort	14:36:55	14:37:05	0000000011111111
52617	/dcs/vlsi/junwei/a4/arms/jacobi	14:37:05	14:37:11	0111111111111111
52624	/dcs/vlsi/junwei/a4/arms/improc	14:37:11	14:37:31	0000000011111111
52633	/dcs/vlsi/junwei/a4/arms/sweep3d	14:37:13	14:37:28	1111111100000000
52663	/dcs/vlsi/junwei/a4/arms/memsort	14:37:43	14:37:53	0000000011111111
52669	/dcs/vlsi/junwei/a4/arms/memsort	14:37:49	14:37:59	1111111100000000
52680	/dcs/vlsi/junwei/a4/arms/fft	14:38:00	14:38:10	1111111111111111
52706	/dcs/vlsi/junwei/a4/arms/closure	14:38:26	14:38:28	0111111111111111
52716	/dcs/vlsi/junwei/a4/arms/closure	14:38:36	14:38:38	0111111111111111
52722	/dcs/vlsi/junwei/a4/arms/memsort	14:38:42	14:38:52	0000000011111111
52738	/dcs/vlsi/junwei/a4/arms/cpi	14:38:58	14:39:00	0000111111111111
52761	/dcs/vlsi/junwei/a4/arms/memsort	14:39:21	14:39:31	0000000011111111
52800	/dcs/vlsi/junwei/a4/arms/closure	14:40:00	14:40:02	0111111111111111
52809	/dcs/vlsi/junwei/a4/arms/closure	14:40:09	14:40:11	0111111111111111
52832	/dcs/vlsi/junwei/a4/arms/jacobi	14:40:32	14:40:38	0111111111111111



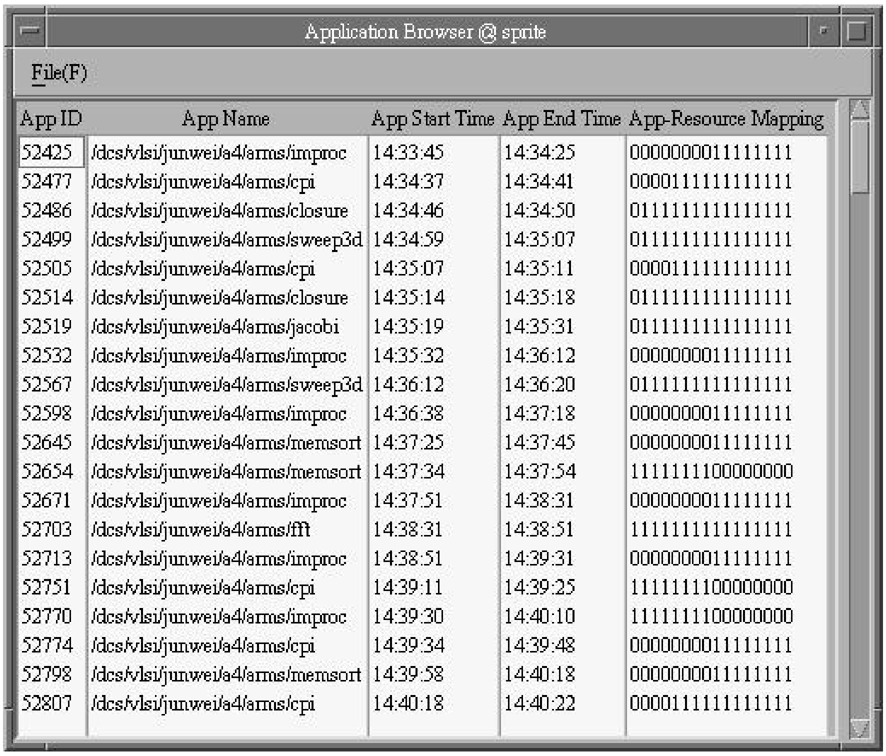
B.3 Experiment Results @ origin



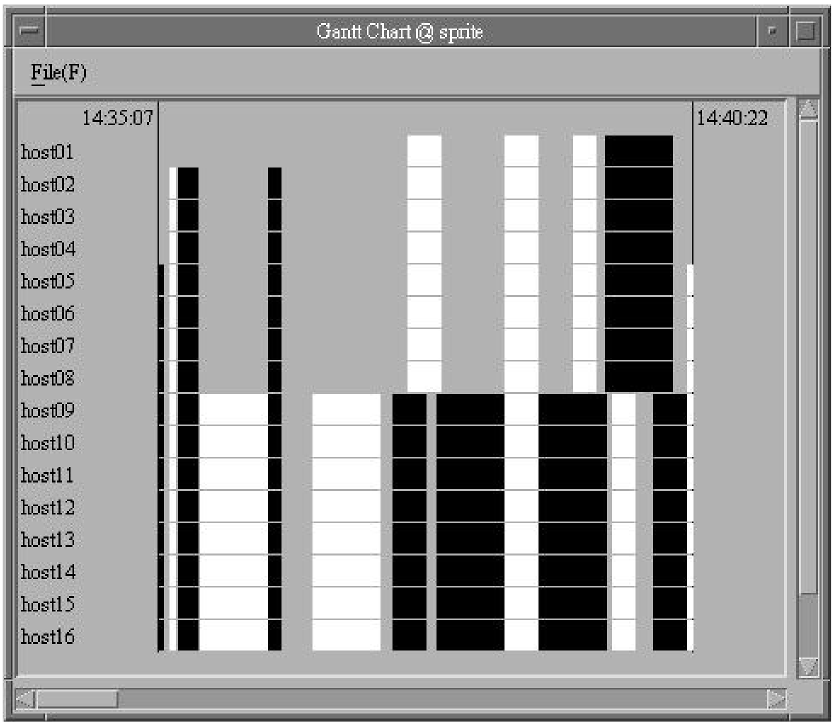
App ID	App Name	App Start Time	App End Time	App-Resource Mapping
52420	/dcs/vlsi/junwei/a4/arms/memsort	14:33:40	14:33:50	0000000011111111
52433	/dcs/vlsi/junwei/a4/arms/cpi	14:33:53	14:33:55	0000111111111111
52451	/dcs/vlsi/junwei/a4/arms/jacobi	14:34:11	14:34:17	0111111111111111
52484	/dcs/vlsi/junwei/a4/arms/jacobi	14:34:44	14:34:50	0111111111111111
52500	/dcs/vlsi/junwei/a4/arms/jacobi	14:35:00	14:35:06	0111111111111111
52593	/dcs/vlsi/junwei/a4/arms/cpi	14:36:33	14:36:35	0000111111111111
52648	/dcs/vlsi/junwei/a4/arms/improc	14:37:28	14:37:48	0000000011111111
52658	/dcs/vlsi/junwei/a4/arms/sweep3d	14:37:48	14:37:52	0111111111111111
52664	/dcs/vlsi/junwei/a4/arms/fft	14:37:52	14:38:02	1111111111111111
52688	/dcs/vlsi/junwei/a4/arms/fft	14:38:08	14:38:18	1111111111111111
52702	/dcs/vlsi/junwei/a4/arms/improc	14:38:22	14:38:42	0000000011111111
52709	/dcs/vlsi/junwei/a4/arms/jacobi	14:38:29	14:38:44	1111111100000000
52718	/dcs/vlsi/junwei/a4/arms/sweep3d	14:38:44	14:38:48	0111111111111111
52729	/dcs/vlsi/junwei/a4/arms/closure	14:38:49	14:38:51	0111111111111111
52775	/dcs/vlsi/junwei/a4/arms/memsort	14:39:35	14:39:45	0000000011111111



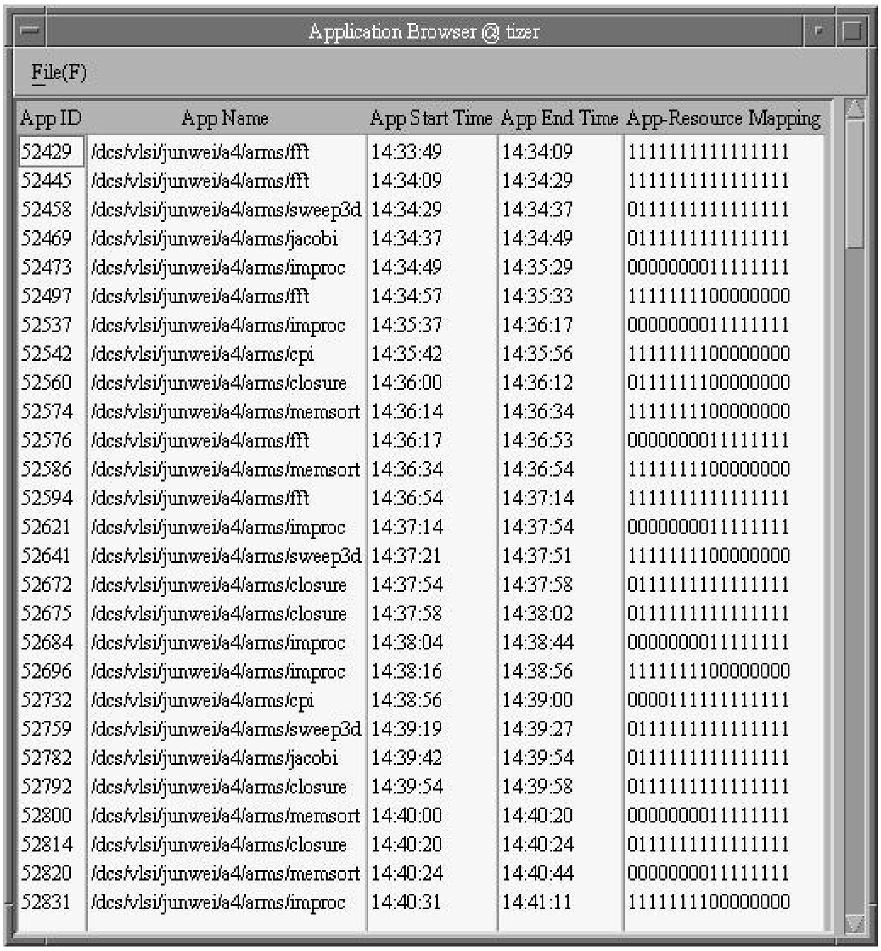
B.4 Experiment Results @ sprite



App ID	App Name	App Start Time	App End Time	App-Resource Mapping
52425	/dcs/vlsi/junwei/a4/arms/improc	14:33:45	14:34:25	0000000011111111
52477	/dcs/vlsi/junwei/a4/arms/cpi	14:34:37	14:34:41	0000111111111111
52486	/dcs/vlsi/junwei/a4/arms/closure	14:34:46	14:34:50	0111111111111111
52499	/dcs/vlsi/junwei/a4/arms/sweep3d	14:34:59	14:35:07	0111111111111111
52505	/dcs/vlsi/junwei/a4/arms/cpi	14:35:07	14:35:11	0000111111111111
52514	/dcs/vlsi/junwei/a4/arms/closure	14:35:14	14:35:18	0111111111111111
52519	/dcs/vlsi/junwei/a4/arms/jacobi	14:35:19	14:35:31	0111111111111111
52532	/dcs/vlsi/junwei/a4/arms/improc	14:35:32	14:36:12	0000000011111111
52567	/dcs/vlsi/junwei/a4/arms/sweep3d	14:36:12	14:36:20	0111111111111111
52598	/dcs/vlsi/junwei/a4/arms/improc	14:36:38	14:37:18	0000000011111111
52645	/dcs/vlsi/junwei/a4/arms/memsort	14:37:25	14:37:45	0000000011111111
52654	/dcs/vlsi/junwei/a4/arms/memsort	14:37:34	14:37:54	1111111100000000
52671	/dcs/vlsi/junwei/a4/arms/improc	14:37:51	14:38:31	0000000011111111
52703	/dcs/vlsi/junwei/a4/arms/fft	14:38:31	14:38:51	1111111111111111
52713	/dcs/vlsi/junwei/a4/arms/improc	14:38:51	14:39:31	0000000011111111
52751	/dcs/vlsi/junwei/a4/arms/cpi	14:39:11	14:39:25	1111111100000000
52770	/dcs/vlsi/junwei/a4/arms/improc	14:39:30	14:40:10	1111111100000000
52774	/dcs/vlsi/junwei/a4/arms/cpi	14:39:34	14:39:48	0000000011111111
52798	/dcs/vlsi/junwei/a4/arms/memsort	14:39:58	14:40:18	0000000011111111
52807	/dcs/vlsi/junwei/a4/arms/cpi	14:40:18	14:40:22	0000111111111111



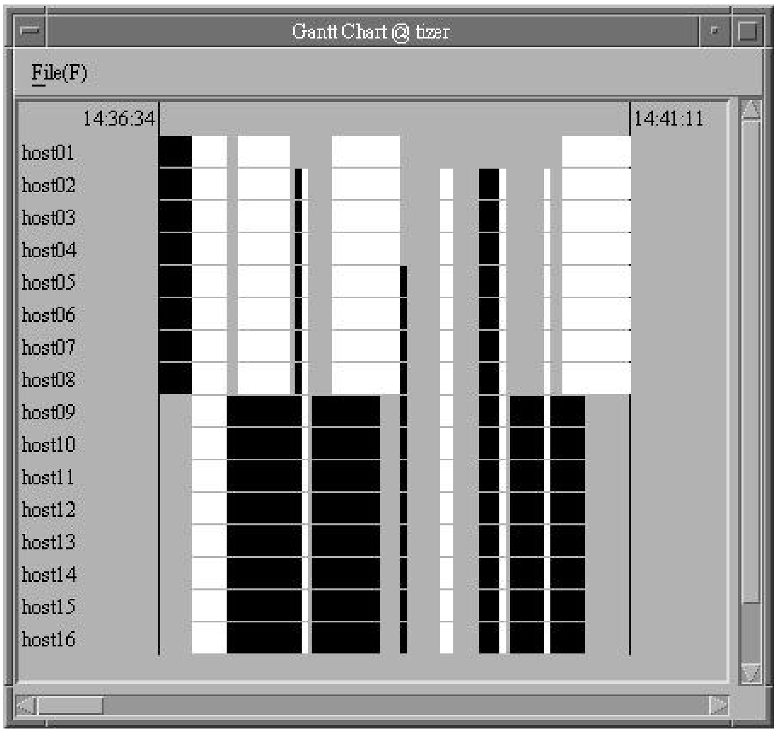
B.5 Experiment Results @ tizer



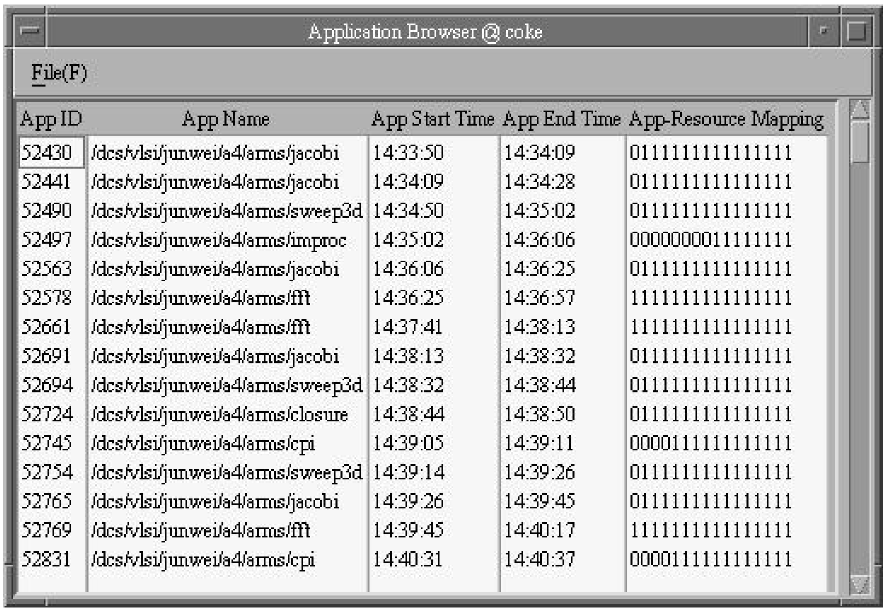
Application Browser @ tizer

File(F)

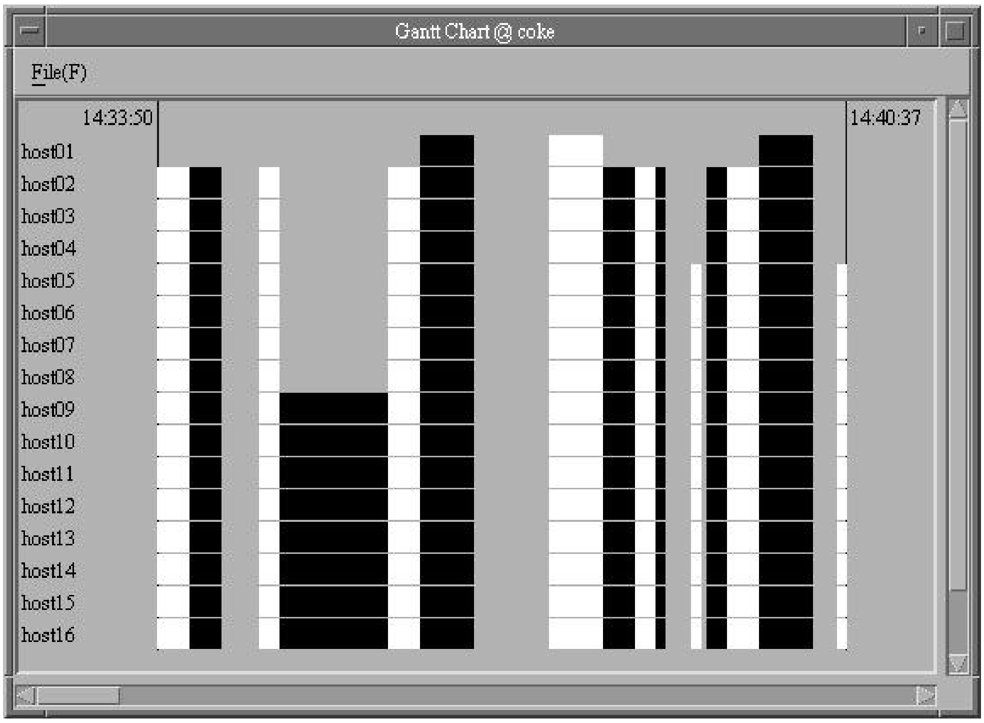
App ID	App Name	App Start Time	App End Time	App-Resource Mapping
52429	/dcs/vlsi/junwei/a4/arms/fft	14:33:49	14:34:09	1111111111111111
52445	/dcs/vlsi/junwei/a4/arms/fft	14:34:09	14:34:29	1111111111111111
52458	/dcs/vlsi/junwei/a4/arms/sweep3d	14:34:29	14:34:37	0111111111111111
52469	/dcs/vlsi/junwei/a4/arms/jacobi	14:34:37	14:34:49	0111111111111111
52473	/dcs/vlsi/junwei/a4/arms/improc	14:34:49	14:35:29	0000000011111111
52497	/dcs/vlsi/junwei/a4/arms/fft	14:34:57	14:35:33	1111111100000000
52537	/dcs/vlsi/junwei/a4/arms/improc	14:35:37	14:36:17	0000000011111111
52542	/dcs/vlsi/junwei/a4/arms/cpi	14:35:42	14:35:56	1111111100000000
52560	/dcs/vlsi/junwei/a4/arms/closure	14:36:00	14:36:12	0111111100000000
52574	/dcs/vlsi/junwei/a4/arms/memsort	14:36:14	14:36:34	1111111100000000
52576	/dcs/vlsi/junwei/a4/arms/fft	14:36:17	14:36:53	0000000011111111
52586	/dcs/vlsi/junwei/a4/arms/memsort	14:36:34	14:36:54	1111111100000000
52594	/dcs/vlsi/junwei/a4/arms/fft	14:36:54	14:37:14	1111111111111111
52621	/dcs/vlsi/junwei/a4/arms/improc	14:37:14	14:37:54	0000000011111111
52641	/dcs/vlsi/junwei/a4/arms/sweep3d	14:37:21	14:37:51	1111111100000000
52672	/dcs/vlsi/junwei/a4/arms/closure	14:37:54	14:37:58	0111111111111111
52675	/dcs/vlsi/junwei/a4/arms/closure	14:37:58	14:38:02	0111111111111111
52684	/dcs/vlsi/junwei/a4/arms/improc	14:38:04	14:38:44	0000000011111111
52696	/dcs/vlsi/junwei/a4/arms/improc	14:38:16	14:38:56	1111111100000000
52732	/dcs/vlsi/junwei/a4/arms/cpi	14:38:56	14:39:00	0000111111111111
52759	/dcs/vlsi/junwei/a4/arms/sweep3d	14:39:19	14:39:27	0111111111111111
52782	/dcs/vlsi/junwei/a4/arms/jacobi	14:39:42	14:39:54	0111111111111111
52792	/dcs/vlsi/junwei/a4/arms/closure	14:39:54	14:39:58	0111111111111111
52800	/dcs/vlsi/junwei/a4/arms/memsort	14:40:00	14:40:20	0000000011111111
52814	/dcs/vlsi/junwei/a4/arms/closure	14:40:20	14:40:24	0111111111111111
52820	/dcs/vlsi/junwei/a4/arms/memsort	14:40:24	14:40:44	0000000011111111
52831	/dcs/vlsi/junwei/a4/arms/improc	14:40:31	14:41:11	1111111100000000



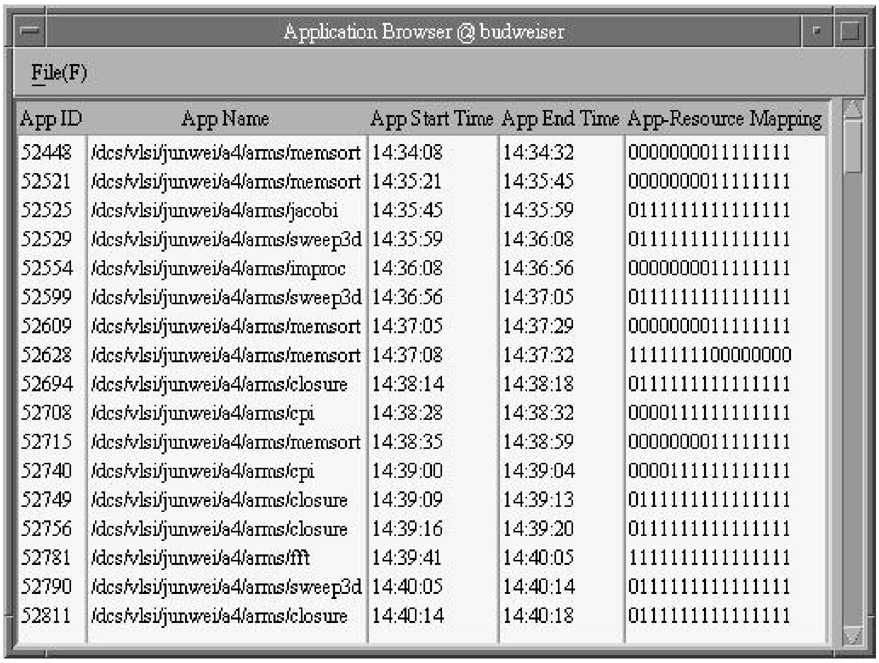
B.6 Experiment Results @ coke



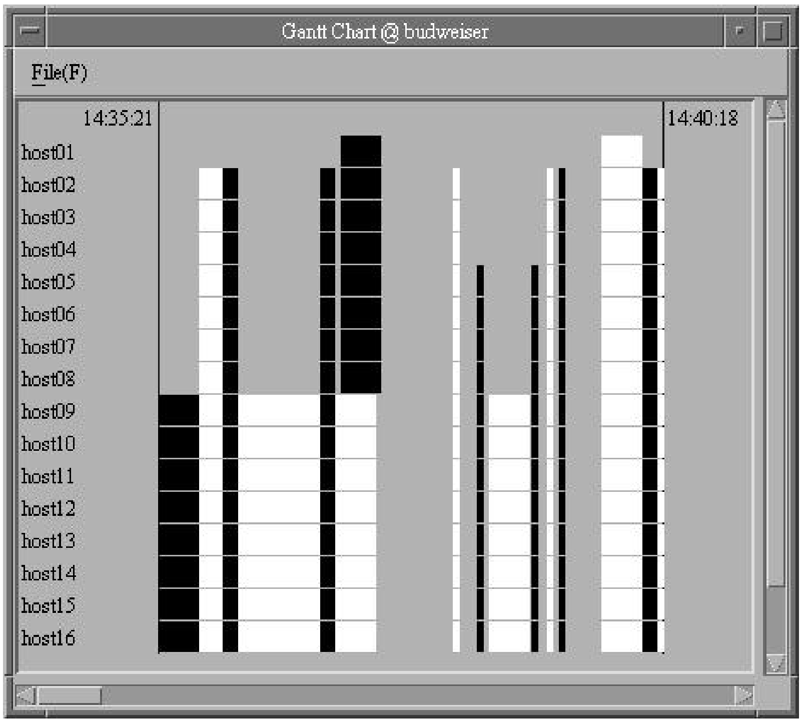
App ID	App Name	App Start Time	App End Time	App-Resource Mapping
52430	/dcs/vlsi/junwei/a4/arms/jacobi	14:33:50	14:34:09	0111111111111111
52441	/dcs/vlsi/junwei/a4/arms/jacobi	14:34:09	14:34:28	0111111111111111
52490	/dcs/vlsi/junwei/a4/arms/sweep3d	14:34:50	14:35:02	0111111111111111
52497	/dcs/vlsi/junwei/a4/arms/improc	14:35:02	14:36:06	0000000011111111
52563	/dcs/vlsi/junwei/a4/arms/jacobi	14:36:06	14:36:25	0111111111111111
52578	/dcs/vlsi/junwei/a4/arms/fft	14:36:25	14:36:57	1111111111111111
52661	/dcs/vlsi/junwei/a4/arms/fft	14:37:41	14:38:13	1111111111111111
52691	/dcs/vlsi/junwei/a4/arms/jacobi	14:38:13	14:38:32	0111111111111111
52694	/dcs/vlsi/junwei/a4/arms/sweep3d	14:38:32	14:38:44	0111111111111111
52724	/dcs/vlsi/junwei/a4/arms/closure	14:38:44	14:38:50	0111111111111111
52745	/dcs/vlsi/junwei/a4/arms/cpi	14:39:05	14:39:11	0000111111111111
52754	/dcs/vlsi/junwei/a4/arms/sweep3d	14:39:14	14:39:26	0111111111111111
52765	/dcs/vlsi/junwei/a4/arms/jacobi	14:39:26	14:39:45	0111111111111111
52769	/dcs/vlsi/junwei/a4/arms/fft	14:39:45	14:40:17	1111111111111111
52831	/dcs/vlsi/junwei/a4/arms/cpi	14:40:31	14:40:37	0000111111111111



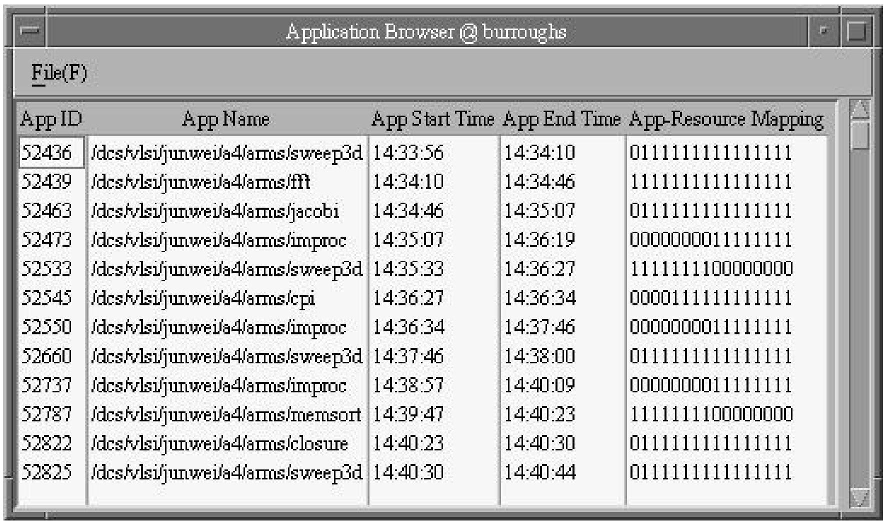
B.7 Experiment Results @ budweiser



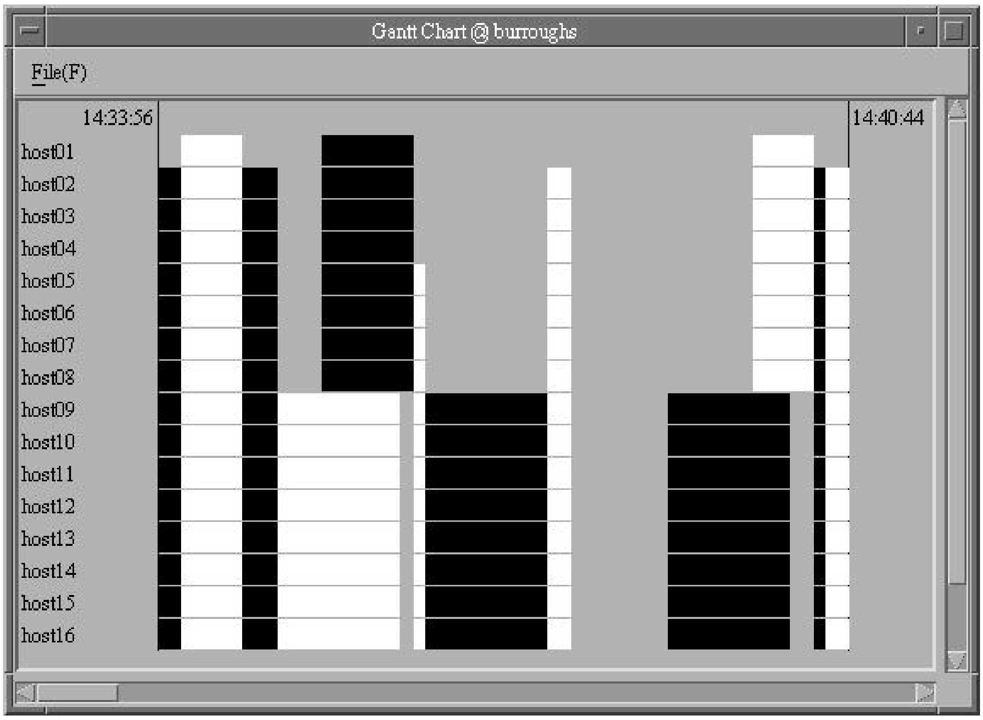
App ID	App Name	App Start Time	App End Time	App-Resource Mapping
52448	/dcs/vlsi/junwei/a4/arms/memsort	14:34:08	14:34:32	0000000011111111
52521	/dcs/vlsi/junwei/a4/arms/memsort	14:35:21	14:35:45	0000000011111111
52525	/dcs/vlsi/junwei/a4/arms/jacobi	14:35:45	14:35:59	0111111111111111
52529	/dcs/vlsi/junwei/a4/arms/sweep3d	14:35:59	14:36:08	0111111111111111
52554	/dcs/vlsi/junwei/a4/arms/improc	14:36:08	14:36:56	0000000011111111
52599	/dcs/vlsi/junwei/a4/arms/sweep3d	14:36:56	14:37:05	0111111111111111
52609	/dcs/vlsi/junwei/a4/arms/memsort	14:37:05	14:37:29	0000000011111111
52628	/dcs/vlsi/junwei/a4/arms/memsort	14:37:08	14:37:32	1111111100000000
52694	/dcs/vlsi/junwei/a4/arms/closure	14:38:14	14:38:18	0111111111111111
52708	/dcs/vlsi/junwei/a4/arms/cpi	14:38:28	14:38:32	0000111111111111
52715	/dcs/vlsi/junwei/a4/arms/memsort	14:38:35	14:38:59	0000000011111111
52740	/dcs/vlsi/junwei/a4/arms/cpi	14:39:00	14:39:04	0000111111111111
52749	/dcs/vlsi/junwei/a4/arms/closure	14:39:09	14:39:13	0111111111111111
52756	/dcs/vlsi/junwei/a4/arms/closure	14:39:16	14:39:20	0111111111111111
52781	/dcs/vlsi/junwei/a4/arms/fft	14:39:41	14:40:05	1111111111111111
52790	/dcs/vlsi/junwei/a4/arms/sweep3d	14:40:05	14:40:14	0111111111111111
52811	/dcs/vlsi/junwei/a4/arms/closure	14:40:14	14:40:18	0111111111111111



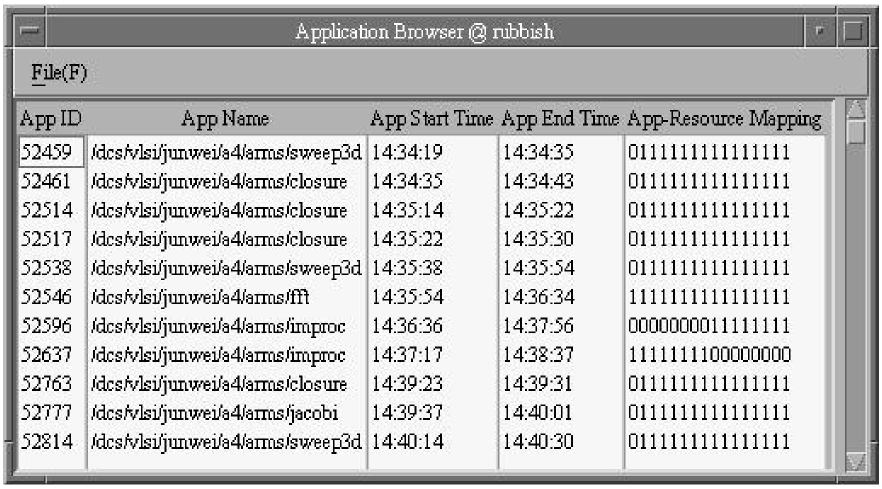
B.8 Experiment Results @ burroughs



App ID	App Name	App Start Time	App End Time	App-Resource Mapping
52436	/dcs/vlsi/junwei/a4/arms/sweep3d	14:33:56	14:34:10	0111111111111111
52439	/dcs/vlsi/junwei/a4/arms/fft	14:34:10	14:34:46	1111111111111111
52463	/dcs/vlsi/junwei/a4/arms/jacobi	14:34:46	14:35:07	0111111111111111
52473	/dcs/vlsi/junwei/a4/arms/improc	14:35:07	14:36:19	0000000011111111
52533	/dcs/vlsi/junwei/a4/arms/sweep3d	14:35:33	14:36:27	1111111100000000
52545	/dcs/vlsi/junwei/a4/arms/cpi	14:36:27	14:36:34	0000111111111111
52550	/dcs/vlsi/junwei/a4/arms/improc	14:36:34	14:37:46	0000000011111111
52660	/dcs/vlsi/junwei/a4/arms/sweep3d	14:37:46	14:38:00	0111111111111111
52737	/dcs/vlsi/junwei/a4/arms/improc	14:38:57	14:40:09	0000000011111111
52787	/dcs/vlsi/junwei/a4/arms/memsort	14:39:47	14:40:23	1111111100000000
52822	/dcs/vlsi/junwei/a4/arms/closure	14:40:23	14:40:30	0111111111111111
52825	/dcs/vlsi/junwei/a4/arms/sweep3d	14:40:30	14:40:44	0111111111111111



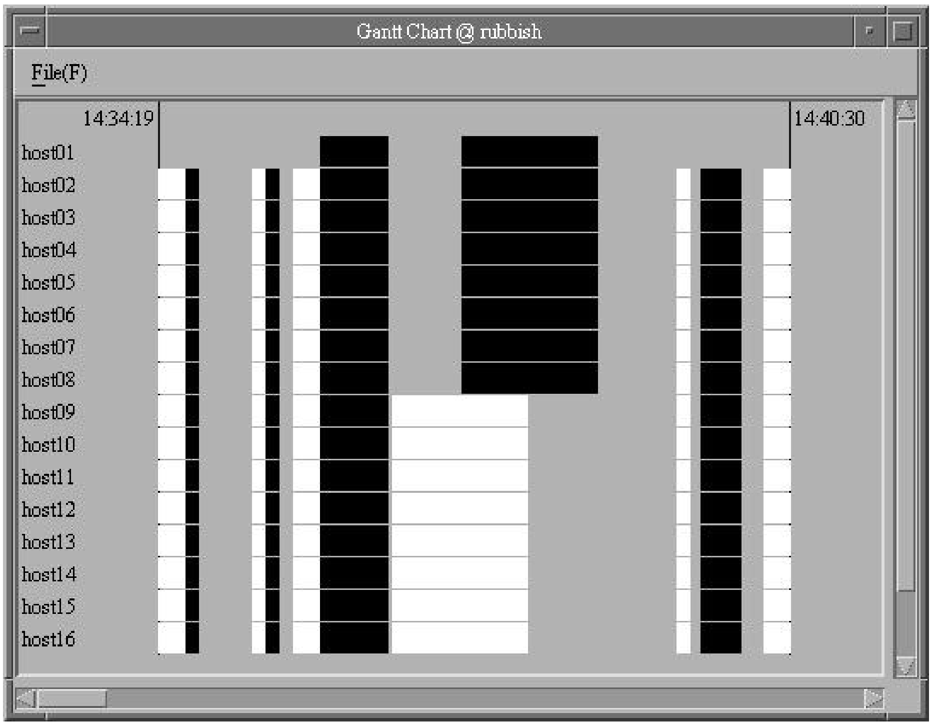
B.9 Experiment Results @ rubbish



Application Browser @ rubbish

File(F)

App ID	App Name	App Start Time	App End Time	App-Resource Mapping
52459	/dcs/vlsi/junwei/a4/arms/sweep3d	14:34:19	14:34:35	0111111111111111
52461	/dcs/vlsi/junwei/a4/arms/closure	14:34:35	14:34:43	0111111111111111
52514	/dcs/vlsi/junwei/a4/arms/closure	14:35:14	14:35:22	0111111111111111
52517	/dcs/vlsi/junwei/a4/arms/closure	14:35:22	14:35:30	0111111111111111
52538	/dcs/vlsi/junwei/a4/arms/sweep3d	14:35:38	14:35:54	0111111111111111
52546	/dcs/vlsi/junwei/a4/arms/fft	14:35:54	14:36:34	1111111111111111
52596	/dcs/vlsi/junwei/a4/arms/improc	14:36:36	14:37:56	0000000011111111
52637	/dcs/vlsi/junwei/a4/arms/improc	14:37:17	14:38:37	1111111100000000
52763	/dcs/vlsi/junwei/a4/arms/closure	14:39:23	14:39:31	0111111111111111
52777	/dcs/vlsi/junwei/a4/arms/jacobi	14:39:37	14:40:01	0111111111111111
52814	/dcs/vlsi/junwei/a4/arms/sweep3d	14:40:14	14:40:30	0111111111111111


Gantt Chart @ rubbish
File(F)
14:34:19
14:40:30
host01
host02
host03
host04
host05
host06
host07
host08
host09
host10
host11
host12
host13
host14
host15
host16