Application Characterisation using a Lightweight Transaction Model

D.P. Spooner, J.D. Turner, J. Cao, S.A. Jarvis, G.R. Nudd *

June 25, 2001

Abstract

With the emergence of GRID environments and distributed systems with dynamic resources and varying user profiles, there is an increasing need to develop reliable tools that can effectively coordinate the requirements of an application with available computing resources. The ability to predict the behaviour of complex aggregated systems under dynamically changing workloads is particularly desirable, leading to effective resource usage and optimisation of networked systems.

In an effort to explore these issues, the High Performance Systems Group (HPSG) at Warwick are developing a resource management architecture that builds upon an existing set of research tools established in the area of performance prediction. The system embodies the concept of a *transaction* that encapsulates a key application component. Using application sensors, user requirements, system policies and resource usage histories, an application is rapidly characterised and scheduled appropriately. A distributed agent-based environment is used as the system infrastructure, and scheduling optimisation is achieved by use of a genetic algorithm whose aim it is to minimise the make-span across systems.

Keywords: *Characterisation, Prediction, Scheduling, Performance, GRID.*

1 Introduction

While the relationship between performance and scheduling has been explored for a number of years [5], interest in geographically dispersed networks such as GRIDs [10, 11] have stimulated demand for high performance resource allocation services. In these new environments, distributed systems must become more adaptive in order to respond to the variations in user demands and resource availability.

*High Performance Systems Group, Dept. of Computer Science, University of Warwick, Coventry. Email: {dps, jdt, junwei, saj, grn}@dcs.warwick.ac.uk

Proceedings of 19th Annual UK Performance Engineering Workshop (UKPEW '01) Leeds, UK, July 2001, pp 215–225.

Research is being undertaken by the HPSG at Warwick to develop tools that assist with issues of resource allocation. This includes the development of a distributed resource management system that is able to predict the behaviour of an application code *prior* to execution and be able to use this information to optimise scheduling. This is achieved by combining an application with a *stub* that includes a map of transactions and their communication, quality of service requirements, system policy filters and historical data pertaining to previous invocations.

This research utilises the A4 [9] agent system, which manages the discovery and advertisement of networked resources. The data collected by the agents is stored persistently in tables which are accessible to local and remote processes. This is currently achieved through an SNMP [16] agent, although it is envisaged that an LDAP [15] infrastructure or emerging GRID standard may be utilised at a later date. The framework also integrates a lightweight version of the group's Performance Analysis and Characterisation Environment (PACE) [1, 2] to rapidly characterise application codes and assist scheduling. The ARM [14] standard is used for response measurement, making it is possible to utilise this tool in conjunction with commercial applications for consolidation and reporting of management data.

1.1 Principal Aims

The framework is designed to facilitate resource discovery, application characterisation and task mapping; key considerations of the research include:

- *Lightweight Characterisation:* The model that describes the resources and the applications is lightweight. Applications are described as a series of transactions that represent particular sub-tasks that are stored persistently by the agents, allowing a broad class of applications to be modeled (including Web services, SQL queries and scientific applications etc.).

- *Metric-Based Scheduling:* While there are existing tools to predict the execution time for a particular sub-task, factors such as the communication requirements, consumption of resources and level of service offered to the user are key components in a modern strategic scheduler. To this end, the environment utilises a general goal-orientated scheduler that can respond to a variety of requirements.

- *Rapid Evaluation Engine:* Evaluation of an application and subsequent mapping to a resource should not incur a significant impact as to reduce the overall efficiency. Finding the optimum solution in a large environment is likely to be expensive and by the time the resource has been located it may have been quicker to go with a lesser solution that was found more rapidly.

- *Dynamic Capabilities:* In a distributed environment consisting of heterogeneous systems, load and configuration are integral factors when making scheduling decisions. While certain parameters can be idealised, the agents maintain system state tables so that scheduling can be prioritised under differing workloads.

- *Scalable Architecture:* As an architecture of this nature can be become potentially large, it is unreasonable that a centralised system could identify and utilise all processing nodes from a single location. It is therefore important that the architecture be scaled in federated hierarchical form to encompass large and transitory infrastructures.

2 Architecture

2.1 Behaviour Prediction using Transactions

Within the framework, applications are composed as a series of transactions that are executed asynchronously or in parallel. A transaction can be considered as a macro-instruction that describes a discrete item of work that possesses a set of platform-independent costs. The *transaction map* contained within an application *stub* provides an indication of how transactions will flow, operate and interact. Agents utilise these maps together with historical information (previous execution timings of similar class transactions) and current system information (load, configuration, etc) to formulate run-time strategies. The agents endeavour to locate suitable candidate systems upon which a task can be allocated and then schedule the task accordingly (see Figure 1).

Each transaction is designated as a particular class of work to allow applications to be classified rapidly. This can be based on technical criterion such as a communication activity, a computation type, or a business classification. The agents associate the application's current behaviour with a transaction defined by the transaction map through use of software, operating system and network sensors.

Applications scheduled by the environment are supplied with an instantiation of a *stub* that initially contains a limited set of static performance information. As the program is executed, the *stub* is populated with dynamic runtime information, which is retained by the scheduling agents to allow subsequent applications of the same class to be scheduled more efficiently.

2.2 Lightweight PACE

The use of transactions as macro-instructions is a direct development of the group's original PACE system, which is designed to predict the expected execution time for scientific applications running on multiprocessor and distributed environments. PACE operates by constructing a theoretical time-line for a given

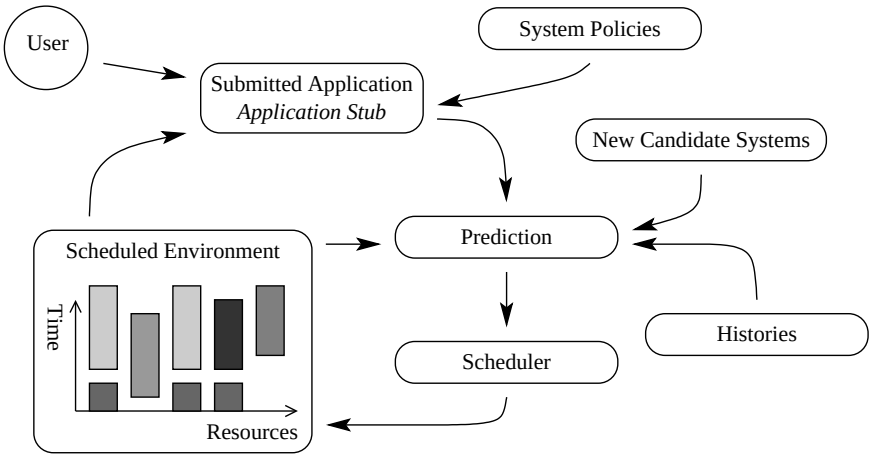

Figure 1: An application is presented to the framework with a *stub* which allocates the process to the required resource based on current usage, system policies and histories. As the code is executed, instrumentation sensors tie activity with the *transaction map*, populating the *stub* accordingly.

application based upon the timings of particular machine code instructions and their organisation as subtasks. The resultant model is parameterised so that factors such as input data size, number of processors, and available memory can be adjusted to observe the effect on execution time.

Adopting a layered approach, PACE associates each layer with a set of specified interfaces to allow objects of the same class to be exchanged without affecting the overall structure of the model. The sequential tasks, the parallelisation strategy and the hardware elements of the given application are described with a modeling language known as CHIP^{3}S[3]. This is subsequently compiled and linked to form a single application object which is evaluated to obtain the expected execution time.

While the lightweight version of PACE (known as PACE Lite) can also generate parameter-based prediction models, the decision to utilise transactions, as opposed to individual machine level instructions permits the system to characterise a wider breadth of applications. As Figure 2 illustrates, the structure of PACE Lite remains similar to that of the original PACE model.

By demarcating applications as *transactions* that encapsulate the key components of an application code, and storing a *transaction map* that describes how the transactions interrelate, the scheduler is able to make performance estimations from previous costs when an application, or parts of an application, are presented subsequently to the system.

It is this combination of run-time histories, closed feedback and lightweight

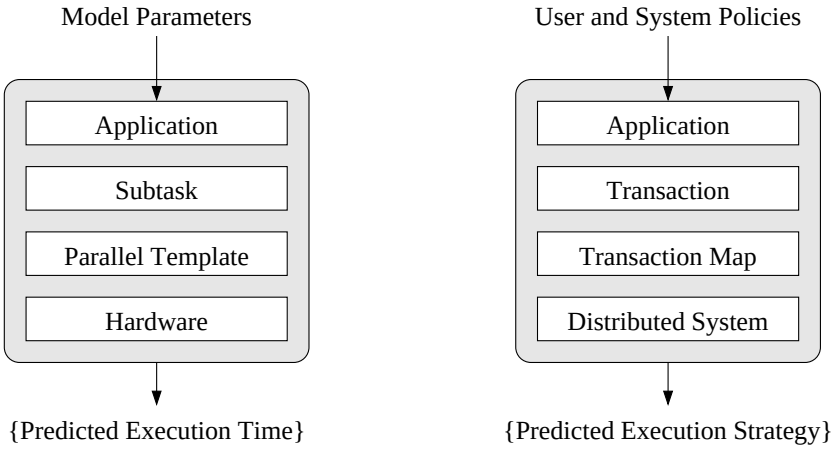

Figure 2: The original PACE structure (depicted on the left) describes applications by means of a layered modeling language. Whilst maintaining the same structure, the lightweight PACE model utilises transactions and maps to characterise applications rapidly.

characterisation, that differentiate this work from the original PACE, and facilitates the characterisation and dynamic scheduling of low workload applications such as web and database systems.

2.3 Agent Hierarchy

Scalability and adaptability are two key challenges when implementing service systems for complex environments such as the GRID. This research utilises the group's A4 model, which identifies a heterogenous group of resources as a hierarchy of homogenous agents that have the capabilities for service provision.

The distributed system component, used as the infrastructure of this research, is illustrated in Figure 3. It consists of a single type of component, the agent, which is used to compose the entire system. Each agent has the same set of functions and capabilities, can send and receive requests, and can provide services to users and other agents.

Each agent provides a scheduler so that applications can be presented to any agent within the hierarchy. The agent will typically 'explore' its surrounding environment in an attempt to locate a suitable resource to run the application. The user is able to specify, by means of the *stub* an upper limit on the exploration. This allows an application with a large workload more scope to locate a suitable candidate system.

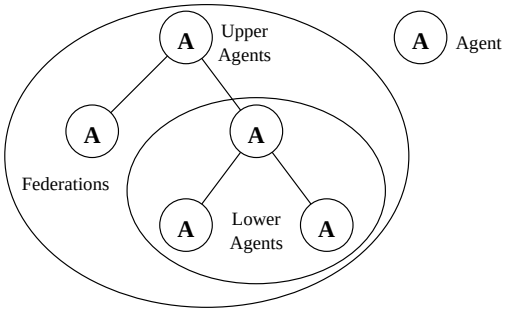

Figure 3: The system adopts a hierarchical agent-based structure. The upper agents act as co-ordinators for the lower agents, although their capabilities remain the same. In this context, any agent can become an upper agent by allowing other agents to register with it.

2.4 Agent Capabilities

The capabilities of the agents are ascribed in modular form as illustrated in Figure 4. The core element is the A4 strategy module which governs the discovery and advertisement of other agents and system resources. The module motivates the agent to federate with lower and upper systems in order to exchange transaction timings and resource information. This assists the agent in fulfilling its high-level objectives of optimising scheduling through cooperation. Scheduling, profiling and costing of application code are provided by various other modules, which are included in the agent system.

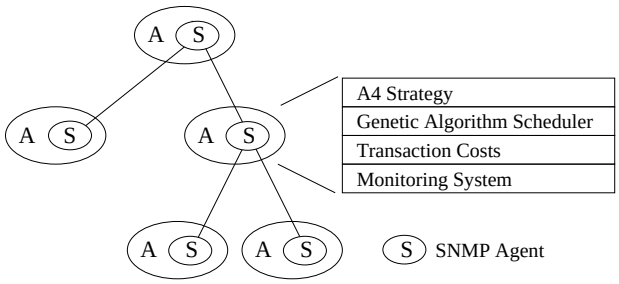

Figure 4: Extensible capabilities of the agents.

The environment retains information regarding resource consumption and timings for each transaction in a table which is accessible by other agents. In the current implementation, this is achieved through the construction of an SNMP manager. The agents have the ability (given sufficient permissions) to

read and write to other agent tables, allowing agents to request and update transaction information. Used in conjunction with a genetic algorithm, this allows the scheduler to improve its capabilities over time.

3 Instrumentation

An integral part of this research is the ability to generate a transaction map and sense the transaction an application is currently executing. Two such methodologies (outlined in Figure 5) are provided. The first method utilises a Java bytecode parser to instrument Java-based applications 'on-the-fly'; the second method monitors the `Java.Net.*` classes in order to tie ends of distributed work together.

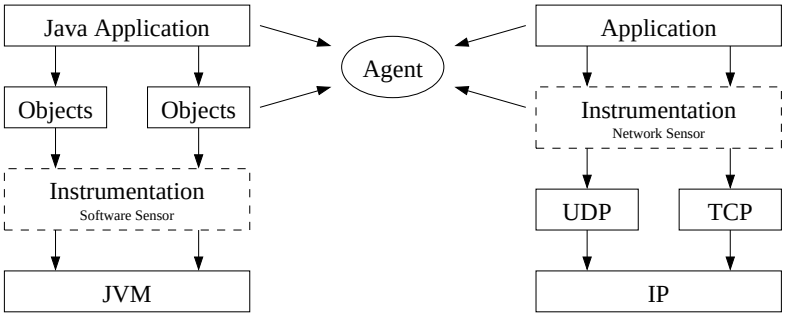

Figure 5: The instrumentation method on the left illustrates software sensing in the Java bytecode. The communication instrumentation, depicted on the right, outlines the capture of transaction activity between the application and network layers.

3.1 ByteCode ARM Instrumentation

An interface has been implemented which allows the environment to instrument Java applications. While a number of tools exist to manipulate bytecode [6, 7, 8], this system automatically identifies 'key' areas of an application and designates them as transactions. The Java bytecode associated with each transaction is then instrumented with ARM transaction method invocations.

The processing and manipulation of Java bytecode is achieved by the implementation of a bytecode parser, written to adhere to the Java Virtual Machine Specification [4]. The parser reads class files and creates a list of objects that describe elements implicit to every Java application. This information can be changed as required and written back to alter the classfile's execution pattern.

ARM (Application Resource Measurement) is an Open Group [12] standard for measuring the performance of defined transactions. It is used to determine

whether transactions meet the performance expectations of the user. With each transaction: a record is kept whether the transaction was a success; the response time of the transaction; and where, when and how many times during the application's runtime this transaction was executed[1].

To ARM an application, it has been necessary in the past for the software developer to define transactions within their software, and to place 'start' and 'stop' calls around each of these transactions before recompiling the code. Using the bytecode parser written as part of this research, it is possible to ARM a Java application automatically (as shown in Figure 6) and obtain the performance statistics of each transaction during runtime.

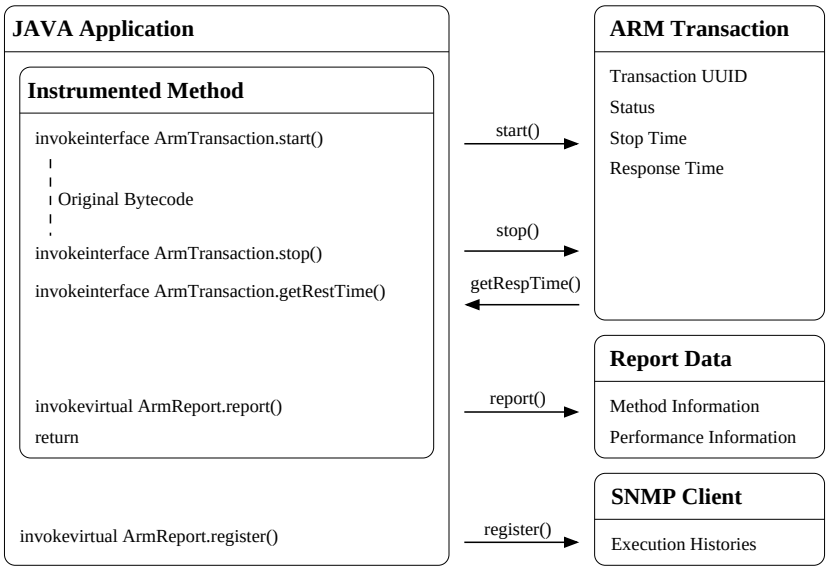

Figure 6: An ARM transaction is started prior to, and stopped after, the method's original bytecode. The response time is then retrieved from the transaction and reported in a *Report Data* object. At the end of the application's execution, all of the reported transaction data is registered in an SNMP table.

The information recorded for each transaction is defined by the consumer implementation of the ARM interface. To date, the lightweight implementation records only execution time; however other ARM consumer implementations for further performance statistics maybe proposed.

The ARM transaction calls allow the environment to obtain transaction performance information from the application during its execution. This information is retained persistently by the appropriate agent and can be used to schedule

[1] Further statistics and information can be found in the ARM (for Java) 3.0 documentation [13].

subsequent applications that consist of similar transactions. The framework benefits from the inherent reuse of objects, and hence by considering applications as a set of transactions, the instrumentation (and subsequent cost analysis) of one application can assist in the instrumentation and scheduling of subsequent applications.

3.2 Communication Analysis

While computation and communication cannot be decoupled entirely, many codes will exhibit predictable and structured messaging regardless of the data content. A highly parallel code, such as a Monte Carlo simulation, is initiated by a master processor that subsequently disperses individual tasks to n slave processes and waits for the responses to fold in.

In this context, the communication behaviour of this particular task will always follow this pattern, regardless of how many processors are available or the data size of the simulation. Hence, there is a level of predictably regarding the application's execution. This behaviour is captured by means of the *transaction map*, in which probabilities with regard to the quantity and locality of data can also be attached. This enables the evaluation system to model different combinations of machines and different network topologies in order to schedule and configure more effectively. Additionally the *stub* can provide an estimation as whether the code will interact with a particular target machine, allowing quality of service targets to be honoured.

4 Conclusions

This paper presents an overview of a lightweight framework for characterising and scheduling distributed applications.

The research builds upon the group's performance and agent work to create a strategic scheduler positioned as GRID middleware. It encompasses the notion of a *transaction* that encapsulates key areas of application code, and captures their behaviour by means of an *transaction map*. Combining prediction, past histories, quality of service requirements and agent-based resource discovery in a *stub*, applications are characterised rapidly and scheduled effectively.

It is envisaged that this system will provide useful insights into building GRID software for widely dispersed and distributed systems.

5 Acknowledgements

The work is funded in part by NASA and USARDSG contract no. 8911-EE-01. The authors would also like to express their gratitude to IBM's T J Watson Research Center, NY. for their contributions towards this research.

References

[1] G.R. Nudd, D.J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper, D.V. Wilcox. PACE - A Toolset for the Performance Prediction of Parallel and Distributed Systems *International Journal of High Performance Computing Applications, Special Issues on Performance Modelling* Vol 14, No 3, pp 228–251 (2000)

[2] J. Cao, D.J. Kerbyson, E. Papaefstathiou, G.R. Nudd. Modelling of ASCI High Performance Applications Using PACE. *Proceedings of 15th Annual UK Performance Engineering Workshop* (UKPEW '99), Bristol, UK, July 1999, pp 413–424.

[3] E. Papaefstathiou, D.J. Kerbyson, G.R. Nudd, T.J. Atherton. An Overview of the CHIP3S Performance Prediction Toolset for Parallel Systems, *8th ISCA Int. Conf on Parallel and Distributed Computing Systems* Florida (1995) pp 527–533.

[4] SUN Microsystems. Java Virtual Machine Specification. `http://java.sun.com/docs/books/vmspec/`

[5] I. Foster, C. Kesselman. The Grid : Blueprint for a New Computing Infrastructure. Morgan Kaufmann, pp 279–290, (1998)

[6] H. Lee, B. Zorn. BIT : Bytecode Instrumenting Tool. University of Washington (1996). `http://www.cs.colorado.edu/~hanlee/BIT/index.html`

[7] M. Dahm. Byte Code Engineering, Freie Univerisitat Berlin. `http://bcel.sourceforge.net/`

[8] G.A. Cohen, J.S. Chase, D.L. Kaminsky. Automatic Program Transformation with JOIE. Technical Paper, `http://www.cs.duke.edu/~gac/joie/`

[9] J. Cao, D.J. Kerbyson, G.R. Nudd. Peformance Evaluation of an Agent-Based Resource Management Infrastructure for GRID Computing. *Proceedings of 1st IEEE/ACM Int. Symposium on Cluster Computing and the Grid* (May 2001), Brisbane, Australia, pp 311–318.

[10] R.J. Allan, J.M. Brooke, F. Costen, M. Westhead. Grid-based High Performance Computing *Technical Report of the UKHEC Collobration* UKHEC (2000). Available from `http://www.ukhec.ac.uk`

[11] W. Leinberger, V. Kumar. Information Power Grid : The New Frontier in Parallel Computing? *IEEE Concurrency* Volume 7, Number 4, October – December 1999.

[12] The Open Group. http://www.opengroup.org

[13] M.W. Johnson, J. Crowe. Measuring the Performance of ARM 3.0 for Java *Technical Report from Tivoli Systems* Available from http://www.cmg.org/regions/cmgarmw/arm30.html

[14] The Open Group. Application Response Measurement (Issue 3.0 - Java Binding). *Open Group Technical Specification* March 2001. Available from http://www.opengroup.org/

[15] T. Howes, M. Smith. LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol. *Macmillan Technical Publishing* (1997).

[16] J. Case, M. Fedor, M. Schpffstall, J. Davin. RFC 1157: SNMP. http://www.faqs.org/rfcs