ELSEVIER

Future Generation Computer Systems 15 (1999) 723–734

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

Predictive resource management for meta-applications

N. Floros, A.J.G. Hey, K.E. Meacham *, J. Papay, M. Surridge

University of Southampton, Parallel Applications Centre, 2 Venture road, Southampton, SO16 7NP, UK

Accepted 14 December 1998

Abstract

This paper defines meta-applications as large, related collections of computational tasks, designed to achieve a specific overall result, running on a (possibly geographically) distributed, non-dedicated meta-computing platform. To carry out such applications in an industrial context, one requires resource management and job scheduling facilities (including capacity planning), to ensure that the application is feasible using the available resources, that each component job will be sent to an appropriate resource, and that everything will finish before the computing resources are needed for other purposes.

This requirement has been addressed by the PAC in three major European collaborative projects: PROMENVIR, TOOLSHED and HPC-VAO, leading to the creation of job scheduling software, in which scheduling is brought together with performance modelling of applications and systems, to provide meta-applications management facilities. This software is described, focusing on the performance modelling approach which was needed to support it.

Early results from this approach are discussed, raising some new issues in performance modelling and software deployment for meta-applications. An indication is given about ongoing work at the PAC designed to overcome current limitations and address these outstanding issues.

Keywords: Resource management; Meta-application; Performance modeling

1. Introduction

Meta-applications may be defined as large, related collections of computational tasks, designed to achieve a specific overall result, running on a (possibly geographically) distributed, non-dedicated meta-computing platform. The PROMENVIR tool is a typical example of such a meta-application, which performs probabilistic analysis of mechanical systems, using Monte Carlo simulations [1]. The meta-application consists of a large number of individual solver runs (e.g. NASTRAN), and the statistical analysis of the collective output results provides insights into the effects of uncertainty on manufacturing properties (e.g. stresses). Other examples of meta-applications are given in Section 2.2.

A meta-computing platform generally consists of a collection of heterogeneous, non-dedicated computing resources, which may include networks of UNIX or NT workstations as well as multiprocessor platforms (e.g. SGI PowerChallenge, IBM SP2). These resources may reside within a company Local Area Network (LAN) or WAN environment (distributed across the Internet).

To carry out meta-applications in an industrial context, one must exploit large-scale, coarse-grain parallelism in order to obtain results in a useful time. Meta-computing offers a way to achieve this, by exploiting idle time on a very large collection of computing resources on a corporate scale. Typically, the application

* Corresponding author

0167-739X/99/$ – see front matter
PII: S0167-739X(99)00022-9

will run during silent hours, in batch-mode, therefore resource management and job scheduling facilities are needed to provide transparent access to the computing resources.

However, to undertake a meta-application, one must also ensure beforehand that it is feasible using the available resources, since no-one can afford to commit their (non-dedicated) computer systems on a corporate scale for an indefinite time. In particular, it is essential that each component task in the meta-application will be sent to an appropriate resource where it can run without running out of disk-space, etc, so that everything will finish within silent hours before the computing systems are needed for other purposes.

This requirement has been addressed by the PAC in three major European collaborative projects: PROMENVIR, TOOLSHED and HPC-VAO, leading to the creation of software system in which job scheduling is brought together with performance modelling of applications and systems to provide meta-application management facilities. The aim of our approach is to exploit models of performance:

- to predict the duration of such applications, so that users can ensure that they fit within the available resources, and
- to ensure that each job is sent to an appropriate host, where it will run sufficiently quickly, and without running out of local resources.

The meta-application management software developed by PAC has been used to run some large-scale applications, including some tests running over the Internet across several secure sites, the results of which are reported elsewhere [2]. The purpose of this paper is to expand on the performance modelling requirements for 'predictive resource management', to show how these have been addressed for a number of application software packages, and to present some enhancements which are the subject of ongoing research at the PAC.

2. Meta-application manager

2.1. Overview

The PAC's meta-application manager incorporates a number of components and features:

- A basic resource management system, incorporating system load monitoring and task execution daemons. This enables the meta-application manager to be used even where no network resource management facilities are available.
- Interfaces to more sophisticated commercial resource management systems: currently LoadLeveler and LSF.
- A resources database, describing the available machines, networks and storage devices, including capacity and performance parameters.
- A graphical system definition and administration interface, enabling the resources data to be defined and updated, and the various daemons to be launched. A library of machine performance parameters is available, so that the system can automatically set these for most machines detected.
- The core meta-application task allocator, which processes descriptions of meta-application task sets, and assigns them to the available resources.
- A database of application performance models, which are used by the task allocator in order to make its decisions.

The software acts as a server, to which meta-applications can connect to exploit its facilities. To enable this, an Application Programming Interface (API) is provided, enabling meta-application developers to build up task graphs, initiate meta-application task allocation processes, extract data on the projected performance at the task or application level, and initiate execution. The meta-application manager does not itself include a graphical interface to these features, since they are typically integrated closely with meta-application itself, but a command-line version of the API is available.

Fig. 1 shows the overall architecture of the meta-application manager, as described above.

2.2. Applications of the meta-application manager

The PAC's meta-application manger software is being used in several research and development projects, whose status is as follows.

The PROMENVIR project [1,3] finished at the end of 1997, having achieved its goal of providing a meta-application software package for probabilistic analysis through simulation of mechanical sys-

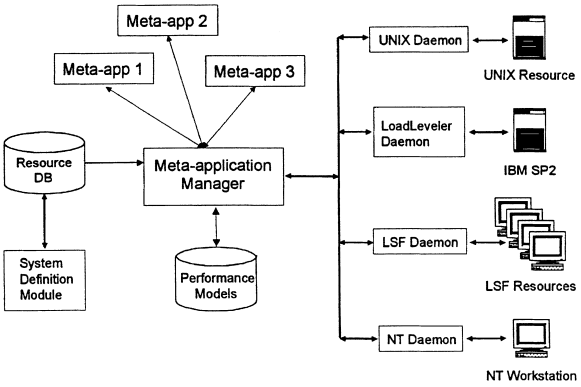


Fig. 1. Meta-application manager architecture.

tems. Typical applications include failure analysis for complex mechanical systems (e.g. space mission components and processes, turbine components, etc) and the effect of uncertainty in such systems (e.g. probabilistic crash analysis, etc).

The PROMENVIR meta-application conducts a Monte-Carlo search of the uncertain parameter space, generating a task-graph consisting of a number of 'shots', each of which consists of a mechanical simulation. PROMENVIR comes with a generic user interface enabling probabilistic features to be defined, based on a single specimen set of simulation data files. A library of probabilistic material parameters and generic distribution functions is available, and sophisticated data processing facilities for statistical analysis of the simulation results.

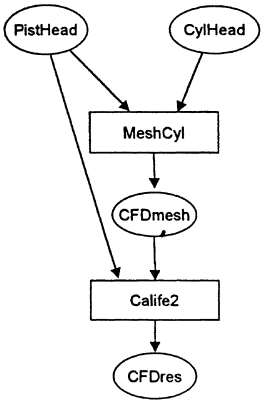
In late the 1997, the meta-application manager was used within PROMENVIR to analyse the behaviour of a satellite antennae deployment, to determine the ways in which the planarity of the antennae depended on uncertainties of manufacture. Deviations from the designed planarity affect the performance of the antennae, and in extreme cases make communication with the satellite impossible, which is obviously critical to satellite operation. The analysis was conducted over the Internet, using a pool of over 100 non-dedicated workstations across eight sites, some of which were secure, one of the first large-scale, industrial meta-applications to be carried out over the Internet in Europe [2].

An earlier version of the PROMENVIR environment, which incorporates some meta-application manager components (although not yet including performance models), is currently distributed commercially through BLUE Engineering in Turin.

The TOOLSHED project [4] is still ongoing and will finish later this year. Toolshed has created a STEP-based parallel simulation environment for commercial mesh-based analysis codes. TOOLSHED incorporates an activity management facility which acts as the meta-application driver, a data-management facility (based on the DEVA software from Rutherford–Appleton Laboratories), a range of mesh generation and partitioning tools (from NUMECA, Bertin and RAL), and data extraction and visualisation (including, run-time visualisation) based on GLView from Sintef/Viewtech. The meta-application manager forms back-end of the activity management facility, which is used to define and manage parametric studies or one-off analysis processes including meshing, simulation and post-processing. Three example applications are being addressed in the TOOLSHED project: a non-linear structural analysis of dams under different load conditions using the PARNASO code from ENEL, an electromagnetic analysis of airframes using the AS-THETIS code from Aerospatiale, and a diesel engine CFD analysis carried out by Ruston Diesels using the CALIFE code from Bertin.

The TOOLSHED system has only recently been installed at the end-user companies, so no industrial test results are yet available. However, demonstrations of the environment with the CALIFE code have been made at Ruston Diesels, in which performance predictions for a previously unseen simulation were accurate to within 15%.

The HPC-VAO project [5] is also ongoing, aiming to provide a framework for vibroacoustic analysis and optimisation and apply it in the automotive sector. The meta-application in this case consists of the optimisation driver Optimus, which is already available from LMS in a non-distributed version. The integration of Optimus with the meta-application manager is still ongoing, so no industrial results are yet available from the end-users in the project (Rover, Renault and BMW). However, a hand-built interface between Optimus and the meta-application manager (based on its command-line interface) has been demonstrated.




```
PRODUCT PistHead
 FILE_LOCATION /usr/calife/test/calife.ted
END PRODUCT

PRODUCT CylHead
 FILE_LOCATION /usr/calife/test/partition
END PRODUCT

PRODUCT CFDmesh
 FILE_LOCATION /usr/calife/test/TOWER.DAT
END PRODUCT

PRODUCT CFDres
 FILE_LOCATION /usr/calife/test/NEW.DAT
END PRODUCT

PRODUCER MeshCyl CALIFE
 COMMAND_LINE_OPTIONS "/usr/calife/calife.exe"
 PERFORMANCEPARAMETERS "-1.5e+02 9e+06 -2.0e+01 6.0e-01 1 0"
 TASKEXECDIR /usr/calife/test
 INPUT_DATASETS "PistHead"
 INPUT_DATASETS "CylHead"
 OUTPUT_DATASETS "CFDmesh"
END PRODUCER

PRODUCER Calife2 CALIFE
 COMMAND_LINE_OPTIONS "/usr/calife/calife.exe"
 PERFORMANCEPARAMETERS "-1.5e+02 9e+06 -2.0e+01 6.0e-01 1 0"
 TASKEXECDIR /usr/calife/test
 INPUT_DATASETS "PistHead"
 INPUT_DATASETS "CFDmesh"
 OUTPUT_DATASETS "CFDres"
END PRODUCER
```

Fig. 2. Example meta-application and resulting task graph.

2.3. Resource allocation strategy

The PAC's meta-application management software assigns component computational tasks to resources, whilst giving feedback to the user, in a series of phases.

Firstly, a meta-application task-graph is received from the driver application. This defines the tasks to be executed, including their input and output datasets, such that the dependencies between tasks (where the output of one forms the input of another) may be determined. In addition, the driver application (or user) can impose constraints on the resources to be used for the task-graph as a whole, or at the individual task level. At present, the number of processors to be used for parallel tasks is entered as a task-level constraint, for example.

Fig. 2 shows a very simple example of a simple meta-application, consisting of two dependent tasks, and a typical task graph file which would be generated (via the manager's API) by the meta-application driver. We define a 'producer' as an individual solver/application program (e.g. NASTRAN), and a 'product' as a file, which may be an input or an output for one or more producers. The dependencies are determined simply from the usage of the defined products, i.e. two tasks are said to be dependent if the output of one is used as the input of the other.

The task graph file is read in by the meta-application manager, then a selection heuristic is applied, based on a comparison of predicted computational loads and storage requirements from different tasks. This defines the order in which the tasks will be considered and assigned to machines, and is designed to ensure that the most expensive or critical tasks are allocated first, to the fastest machines. At present, this selection heuristic is based on a simple analysis of the critical path (based on computational load), ensuring that critical tasks are considered first.

The next phase considers each task in turn, and pre-assigns it to the machine that will return the fastest possible result. At this stage, the meta-application manager applies additional constraints based on the availability of software licenses, and predictions of the memory and disk space needed by the task. The software also maintains a database of task allocations to machines, so that the loads represented by these tasks can be taken into account when allocating subsequent tasks. During the pre-assignment phase, parallel scalability models are consulted, and feedback given to the user on the best number of processors to use (usually the smallest number which would allow the code to run in-core, since this gives the best possible throughput). The predicted execution time of the whole task-graph (and, if desired, individual tasks) is also available through the API, so that the

meta-application user can establish whether the scale of the application defined is feasible. At this stage, the user can abort the meta-application, impose different resource constraints, or negotiate with management for the use of additional resources.

Finally, the user initiates execution. During execution, the meta-application manager keeps track of the progress of each task, and dynamically adjusts the assignment of tasks to machines where appropriate. Typically, at least some tasks do not run on the machine originally assigned, due to inaccuracies in the performance models, and uncertain factors such as unexpected background loads. The software does not support dynamic migration of tasks once they start executing, but seeks to avoid initiating any new task on a machine that becomes inappropriate or unavailable.

3. Performance modelling

3.1. Background

Performance modelling originates from the application of complexity theories to computational systems and applications, especially in the high-performance computing (HPC) regime. Early complexity models of parallel systems such as the PRAM model have been extended in abstract models such as Bulk synchronous Parallelism (BSR) [6] and LogP [7], to include more realistic parallel computing phenomena such as communication bottlenecks and synchronisation. This has led to the possibility of modelling real system architectures and application software, to enable their developers to understand and tune their computational efficiency.

Performance modelling has developed alongside benchmarking and performance monitoring, which also provides information about computational efficiency and effectiveness. The level of sophistication used in all three areas has grown along with the complexity of the systems and software to be modelled. Thus simple benchmarks such as the Livermore Loops [8] have been superseded by more sophisticated benchmarks incorporating models of 'typical' behaviour such as SPEC [9,10], ParkBench [11] and EuroBen [12]. At the same time, performance monitoring techniques have moved beyond simple tools to monitor processing loads, to encompass full event tracing [13], and sophisticated data processing and visualisation tools [14].

Not surprisingly, the most sophisticated approach to performance modelling today, therefore involves constructing a detailed model of the computer at various levels, ranging from the main components of the architecture to the gate level [15]. The performance of an application can be predicted by simulating the detailed behaviour of the system when running the application. This approach is highly rewarding for systems architecture developers, but the simulation models take a long time to execute, and are platform-specific.

A variation known as performance characterisation uses a simplified model of the computing platform, constructed from a common basis set of system architecture component models. The performance of an application is still obtained through simulation, but this can now be generated in a standard way from the application source code, so that it is easy to produce comparative models for different platforms. However, the simulations are still quite slow, and one needs access to source code or detailed descriptions of code behaviour, in order to generate them. A typical example of this approach is given by the PEPS project [16], and more recent derived work such as PACE [17].

Neither of these detailed modelling approaches is practical for on-line, predictive resource management with third-party, commercial application software packages. The simulations needed to produce performance predictions are too slow, and the generation of these simulations requires detailed knowledge of application code structure and system architectures. Furthermore, detailed modelling cannot easily handle the input data-dependence of performance, without going to a level of detail comparable to the application being modelled.

For predictive resource management, one needs a simpler approach in which:

- Performance predictions can be derived rapidly in real-time;
- Account is taken of the effect of the input data on performance;
- Detailed knowledge and representation of code structures and machine architectures is not required; and
- Results should allow performance and resource consumption to be compared easily between different jobs and machines.

At first, this seems like an impossible task, but fortunately, one does not need highly accurate models in order to extract useful information for meta-application management.

3.2. Performance modelling architecture

The PAC meta-application manager uses performance models in two ways:

- To find a machine-independent 'weight' through which to compare different tasks during the selection phase;
- To compare completion times for a specific task on different machines, during the predictive, pre-assignment phase, and also for dynamic re-assignment of tasks during execution of the meta-application.

To support this, a three-tiered performance modelling architecture is used, in which tasks are characterised through:

- Application-specific problem-size parameters, extracted from the input decks;
- Machine-and application-independent loads, obtained from the problem sizes via an application model; and
- Predicted run-time and resource usage on a specific machine, obtained from the loads via an application-independent machine model.

This arrangement, and its use within the meta-application task allocation algorithm is illustrated in Fig. 3.

3.3. Machine model

Within this architecture, the machine model must be independent of the application codes. Furthermore, since the machine model is used repeatedly within the task assignment process, during, as well as prior to, the meta-application execution, it must be computationally lightweight.

For these reasons, a simplistic approach has been adopted in which the application is characterised via the 'computational load' (L_{cpu}), the 'I/O load' (L_{io}), the 'communication load' (L_{comm}), and the memory and disk space needed (V_{mem} and S_{disk}). The machine model then consists only of a set of 'rates' for computation (R_{cpu}), I/O (R_{io}) and communications (R_{comm}), along with the available memory and disk space (S_{mem} and S_{disk}. For parallel codes, performance scalability is modelled via the communication load, and by assuming a simple Amdahl decomposition of the loads (and space requirements) into serial and parallel portions, in which f is the fraction remaining sequential. Finally, for some commercial codes, it has been found necessary to include an overall machine-independent start-up time (T_{start}), to allow for license authentication, lock files, etc., which is presumably installation dependent (since it will depend on the software license server).

The machine model then maps loads to the run-time and utilisation via:

$$T(P) = T_{\text{start}} + \frac{L_{\text{cpu}} f_{\text{cpu}}}{R_{\text{cpu}}} + \frac{L_{\text{cpu}}(1 - f_{\text{cpu}})}{P R_{\text{cpu}}} + \frac{L_{\text{io}} f_{\text{io}}}{R_{\text{io}}} + \frac{L_{\text{io}}(1 - f_{\text{io}})}{P R_{\text{io}}} + \frac{L_{\text{comm}}(P)}{R_{\text{comm}}}$$

and

$$U_x(P) = \frac{L_x(P)}{T(P) R_x}$$

where the load $L_x(P)$ is found directly (in the case of L_{comm}) or via the Amdahl partitioning (for L_{cpu} and L_{io}).

The only remaining problem is to determine the units of measurements for the various loads and space requirements. For everything except L_{cpu}, one can simply measure the volume of data to be transferred or stored in bytes, and use bandwidth figures (bytes/sec) for the corresponding machine rates.

The problem of establishing a representative measure for processing speed is a well researched area, and initially it was hoped to use the well-respected SPECmark figures, which provide a good comparative measure of machine performance. Unfortunately, the standard SPEC-unit has been changed every few years, so that a single comparative set of SPECmark figures is not available for all machines likely to be used. For this reason, the unit of measure for L_{cpu} was taken as the 'flop equivalent', that is the amount of computation which would take the same time as a single floating point calculation. The machine parameter R_{cpu} is measured using the public domain benchmark FLOPS V2.0 [18] which can obviously be executed on any machine encountered.

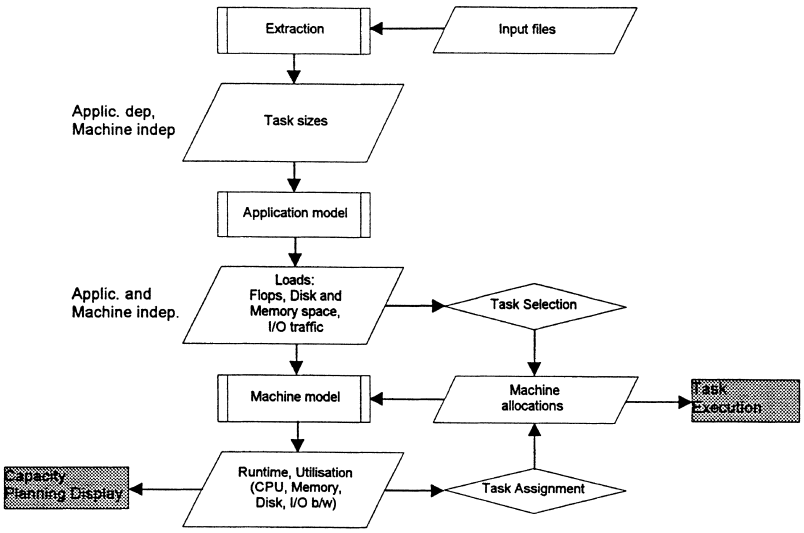


Fig. 3. Performance modelling architecture.

3.4. Application performance characterisation

The application models in this tiered performance modelling architecture consist of relationships between the problem size parameters (extracted from the input files) and the machine-independent loads. These models are obtained empirically by:

- Selecting the (application-specific) problem size parameters to be used, and constructing an extraction utility to obtain them from the input files;
- Running benchmarks for a range of problem sizes, and for parallel codes on different numbers of processors, to obtain measurements of run-time, memory usage, etc.
- Inverting the machine model to obtain the loads corresponding to each benchmark;
- Fitting analytical functions of the problem sizes to account for the observed loads.

Typically, the final step involves choosing the type and order of candidate functions based on a knowledge of the application and its probable algorithms: thus one chooses 3rd-order polynomials for FE codes, for example. However, no access to the source code is needed for this empirical modelling approach.

The final application models obtained through this procedure will be crude, but are able to give a reasonable estimate of which jobs should receive priority in the task allocation procedure. They can also help a user to determine whether a given meta-application will finish in (say) a weekend, albeit with fairly large error bars.

4. Application models

4.1. Overview

Application models have been developed for:

- CALIFE: a CFD application from Bertin et Cie (France);
- MSC/NASTRAN: the statics solution sequence from MacNeal–Schwendler's well-known finite elements code;
- PARNASO: a non-linear structural analysis code from ENEL (Italy);
- AS-THETIS: an electromagnetics analysis code from Aerospatiale (France);
- SIMAID: a multi-body dynamics code with application to robotics, from CEIT (Spain).

Currently, the PAC is developing further models for:

- MSC/NASTRAN: the modal analysis sequence; and
- SYSNOISE: the acoustics analysis code from LMS (Belgium).

Not surprisingly, since many of these applications are commercially available, we are unable, for reasons of commercial sensitivity, to report any details of the performance models obtained. However, we can pro-

vide an outline of the procedure as applied to the (sequential) CALIFE application, as an example of the modelling approach.

4.2. Example: CALIFE

CALIFE is a Computational Fluid Dynamics (CFD) code developed in France by Bertin et Cie. Bertin uses it in their engineering consultancy business, and also by some of their customers including Ruston Diesels, a partner in the TOOLSHED simulation environment project, described in Section 2.2.

The characterisation of CALIFE can be broken into two parts:

- finding the loads per iteration of the CFD time-marching procedure (i.e. loads per time-step);
- finding the rate of progress of this procedure, and hence the number of iterations to be carried out.

A series of benchmarks were provided by Bertin, encompassing a typical internal combustion calculation of the type used by Ruston Diesels, on a range of meshes ranging from 5000 to 160 000 tetrahedral elements.

The inversion of the machine models to provide load information was conducted using Sun and Silicon Graphics benchmarks. However, the SGI version of the code was compiled on a 32-bit (R4000 series) system, so that the normal R_{cpu} value of the PowerChallenge benchmark platform could not be used. This problem had been encountered before, and a rate obtained by compiling the FLOPS20 benchmark in the same way. This gave a reasonably good agreement for loads between the 32-bit and 64-bit machine architectures (see Fig. 4).

It was found through the performance characterisation process that:

- The computational and I/O loads, and also the memory and disk-space requirements are proportional to the number of time-steps, and are linear in the problem size;
- Additional loads are introduced when simulating droplet sprays, in proportion to the number of droplets represented in the simulation at any time;
- The number of time-steps required to complete a run depends on the mesh size, in a way which itself depends on whether droplets are present in the simulation.

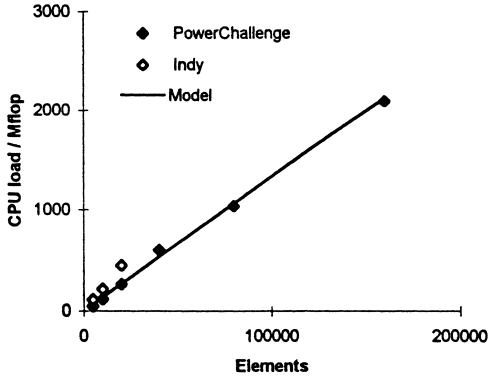


Fig. 4. Relative CPU load in the CALIFE gas compression phase.

The key problem sizes were therefore:

- The simulated simulation duration, both with and without spray (Δt_{gas} and Δt_{spray});
- The number of elements in the mesh, N;
- The rate of droplet injection during the spray phase (droplets per time-step), I; and
- The rotation speed of the engine, r.

The performance model therefore takes a simple form:

$$L_{\text{cpu}} = L^{\text{gas}} + L^{\text{spray}}$$

$$L^{\text{gas}} = n^{\text{gas}}(A + BN)$$

$$L^{\text{spray}} = n^{\text{spray}}\langle n^{\text{drops}}\rangle\beta$$

L_{cpu} is the total computational load, divided into L^{gas} (the load when droplets are not present) and L^{spray} (the load when droplets are present). The parameters A and B represent the start-up computation and the computation per element for the gas flow calculations, and β the computation per droplet in each time-step. The total number of time-steps is given by:

$$n^{\text{gas}} = \frac{\Delta t_{\text{gas}}}{r\delta t_{\text{gas}}}$$

and the number of steps in which spray is represented by:

$$n^{\text{spray}} = \frac{\Delta t_{\text{spray}}}{r\delta t_{\text{spray}}}$$

where the average time-step (δt_{gas}, δt_{spray}) in each case depends on the mesh size according to the following (empirical) relationship:

$$\delta t_{\text{gas}} = aN^{1/2}$$

$$\delta t_{\text{spray}} = b - cN$$

Finally, the average number of drops in the simulation during the spray phase is given by:

$$\langle n^{\text{drops}} \rangle = I\left(\frac{\tau}{\delta t_{\text{spray}}}\right)$$

where τ is the average survival time of a droplet during the simulation.

The parameters A, B, β, a, b, c and τ are determined empirically from benchmark data. The values cannot be given here since CALIFE is a commercially available code, whose precise performance is obviously commercially sensitive. Similar models have been derived for the disk and memory as well as computational resource usage.

This model is evidently very simple, whilst taking account of the input data in a reasonable way, and gives an upper bound on resource needs. When the TOOLSHED environment was installed at Ruston Diesel for the first time, and demonstrated using a test case and execution platform which were not amongst the benchmarks data, the predicted run-time was accurate to within 20%.

4.3. Impact on meta-application management

Run-time predictions are used directly by the meta-application manager for capacity predictions, which help the user ensure that their meta-application will be completed during silent-hours. Clearly, these predictions will be accurate to within 20% at best, so the user should allow a sensible margin for error to ensure that the meta-application does not impact 'daytime' users of the resources.

The accuracy of run-time predictions will affect, to some extent, the resource allocation decisions of the scheduler. However, it is the relative run-times between applications running on different resources which are most important to critical path analysis and resource selection. Furthermore, the initial allocation decisions can be changed dynamically when a task or resource is ready to go, so the overall performance of the system is not very much dependent on the accuracy of the absolute run-time predictions for each task.

Finally, the ability of the meta-application manager to ensure robust, reliable and efficient execution of meta-applications depends most critically on the models of disk and memory usage, and not on the run-time predictions. In practice, space predictions are usually much more deterministic and accurate than time predictions, and it is easier to assign a safety margin, so that the system is able to avoid unpleasant overnight crashes due to lack of space, even where the run-time predictions are relatively crude.

5. Discussion

5.1. Transparent access to meta-computers

The value of the PAC's approach to meta-application management was demonstrated through the PROMENVIR Internet experiment, outlined in Section 2.2 and described in [2]. In this experiment, a typical probabilistic engineering problem from the space industry (developed by CEIT and CASA) was analysed, by running a large stochastic Monte Carlo simulation. A total of 102 CPUs were used (various types of SGI architecture, ranging from R3000 workstations to R10000 multi-processor machines), at eight partner sites, some equipped with firewall security, and all accessed via the Internet. This experiment was therefore representative of industrial meta-applications run within the large manufacturing consortial often found in Europe (e.g. ARIANE and Airbus), and we believe that this represents one of the largest pan-European meta-computing simulations ever carried out.

Each machine within the meta-computer was installed with the SIMAID solver from CEIT [19], and the required meta-application manager daemons. The simulation consisted of 1000 individual (independent) SIMAID runs. Results were sent back to the PROMENVIR driver module, running on an SGI workstation at PAC (along with the meta-application manager), and analysed statistically.

Many lessons were learned from this experience relating to security, Internet reliability and solver version control, which are described in [2]. In the context of this paper, the most interesting lesson related to machine availability: of the 102 machines provided, only 79 were actually used during the experiment. Many of the 'missing' machines were lost due to high background loads (the meta-application manager only dispatches tasks to machines whose load is

below a pre-defined threshold). A few of the slowest machines (R3000 series SGI workstations) were deliberately avoided by the system because their predicted performance was too poor. It is evident that, had jobs been sent to these systems, the run-time for a single shot would have been close to that for the meta-application as a whole, bringing the risk that these shots would have slowed down the whole run!

The ability of our software to factor out machines which are too slow on a case-by-case basis means that one can include and exploit even slow machines (e.g. desktop PCs) in those applications where they can contribute. Clearly, one cannot expect to define a different set of machines across multiple sites for each meta-application, so without performance management features, one would be forced to exclude all older, less powerful system, however numerous.

5.2. Cost and accuracy of performance models

The accuracy of our performance predictions for a hitherto unseen CALIFE test problem is also interesting. Obviously, with the crude approach we have had to employ one could not hope even to be within 50% of the actual run-time for all possible test cases. However, because our model was derived from cases representative of Ruston's expected use of the CALIFE code, we achieved a much more accurate prediction for their problems, as described above.

This raises the question of how many different models might one need for a given application. Clearly, high accuracy can be obtained by selecting a representative set of benchmarks for each user, but the cost of running these benchmarks and deriving a model is high. One cannot expect users to perform such calibration exercises for themselves, and even for an expert, several weeks of effort may be required. The same problem arises if the end-user decides to employ a new code, for which no performance model is available. The PAC has introduced a simple default model for such cases, but this model is designed only to stabilise the task allocation algorithms, and not to produce any performance prediction for unknown codes.

The PAC has recently started a new project, funded by the UK Engineering and Physical Sciences Research Council, to address this problem. The project is now investigating the use of adaptive computing methods (e.g. neuro-fuzzy methods) as a basis for self-calibrating performance models. The concept involves logging performance data for codes in operational use, and using this to refine performance models, which fit the logged data. Since the data come from operational use, they will be representative of the user's typical work load, and if a new application is encountered, a new model fitting process can be initiated.

5.3. Other meta-application issues

The final point that has emerged from our experiences with meta-applications, and attempts to provide management systems for them, is the problem of software access.

Most of the meta-applications being conducted around the world use free or proprietary software, which can be installed on all available platforms at no cost. For example, the PROMENVIR Internet experiment discussed above used the SIMAID application from CEIT, a partner in the PROMENVIR project. Evidently, industrial users want to exploit third-party, commercial software packages, but the cost of installing these on all available machines is high, especially if these machines are distributed throughout a manufacturing consortium.

To address this problem, the PAC is now starting to investigate new software supply paradigms, and this will be the focus for a new ESPRIT project, DISTAL, starting this year.

6. Conclusions

Industrial meta-applications require sensible management facilities, which poses new challenges for performance prediction as well as resource management. The PAC's meta-application management software has addressed some of these, helping to make commercial meta-application driver software available.

Our experience has also raised new issues, which we are addressing through new initiatives investigating on-line, automatic performance characterisation and new commercial software provision paradigms.

References

[1] PROMENVIR web page. http://www.atos-group.de/cae/index-promenvir.htm

[2] K.E. Meacham, N. Floros, M. Surridge, Industrial stochastic simulations on a European meta-computer, in: Proc. EuroPar '98, Southampton.

[3] J. Marczyk, Meta-computing and computational stochastic mechanics, in: J. Marczyk (Ed.), Computational Stochastic Mechanics in a Meta-Computing Perspective, 1997, CIMNE, Barcelona, pp. 1–18.

[4] TOOLSHED project web page. http://www.bertin.fr/~gruez/toolshed.html

[5] HPC-VAO project web page. http://www.lmsintl.com/solutions/

[6] L.G. Valiant, A bridging model for parallel computation, Commun. ACM 33(8) (1990) 103–111.

[7] D. Culler, R. Karp, D. Patterson, LogP: towards a realistic model for parallel Computation, ACM SIGPLAN Notice 28(7) (1993) 1–12.

[8] F.H. McMahon, The Livermore FORTRAN kernels test of the numerical performance range, Performance Evaluation of Supercomputers (1998) 143-186.

[9] K.M. Dixit, The SPEC benchmarks, Parallel Computing 17(10/11) (1991) 1195–1209.

[10] R.P. Wiecker, A detailed look at some popular benchmarks, Parallel Computing 17(10/11) (1991) 1153–1172.

[11] R.W. Hockney, M. Berry, PARKBENCH report: public international benchmarks for parallel computing, Scientific Programming 3(2) (1994) 101–146.

[12] A.J. van der Steen, Status and direction of the EuroBen benchmark, Supercomputer 10(4/5) (1993) 19–31.

[13] P.H. Worley, A new PICL trace file format, ORNL/TM-12125, Oak Ridge National Laboratory, 1992.

[14] D.A. Reed, Scalable performance analysis: the Pablo performance analysis environment, in: Proc. Scalable parallel libraries Conf., IEEE Computer Soc. Press, Silver Spring. MD, 1993, pp. 104–113.

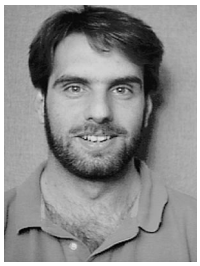
[15] S.K. Reinhardt, M.D. Hill, J.R. Larus, A.R. Lebeck, J.C. Lewis, D.A. Wood, The Wisconsin wind tunnel: virtual prototyping of parallel computers, in: Proc. 1993 ACM SIGMETRICS Conf. on Measurements & Modelling of Computer Systems, 1993, pp. 48–60.

[16] G.R. Nudd, E. Papaefstathiou, J. Papay, T.J. Atherton, C.T. Clarke, D.J. Kerbyson, A.F. Stratton, R. Ziani, M.J. Zemerly, A layered approach to the characterisation of parallel systems for performance prediction, in: Proc. Performance Evaluation of Parallel Systems (PEPS '93), University of Warwick, November 1993, pp. 26–34.

[17] D.J. Kerbyson, E. Papaefstathiou, J.S. Harper, S.C. Perry, G.R. Nudd, Is predictive tracing too late for HPC users, in: R.J. Allan, A. Simpson, D.A. Nicole (Eds.), High Performance Computing, Plenum Press, New York, 1998.

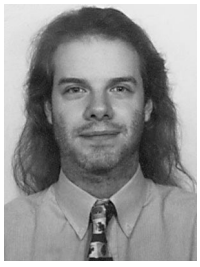
[18] FLOPS2.0 benchmark, incorporating technical description, originally obtained via aburto@marlin.nosc.mil

[19] SIMAID software available from CEIT, San Sebastian. http://www.ceit.es



Nick Floros received a 1st Class Honours degree in Physical Science and computing from South Bank Polytechnic in 1989, then studied for a PhD in High Performance Computational Physics at Southampton University. He joined the Parallel Applications Centre in 1996, where he made a major contribution to developing the Intrepid resource management software. During this time, he worked on two main meta-application projects, HPC-VAO (distributed NVH optimisation) and TOOLSHED (an environment for engineering design analysis).

Tony Hey is Professor of Computation and Head of the Department of Electronics and Computer Science at the University of Southampton. He is also Chairman of the Parallel Applications Centre (PAC), which works closely with industry in a technology transfer role. Professor Hey is on the editorial board of a number of scientific journals and has been on the organising committee of many international conferences. His current research has been concerned with performance estimation for parallel applications and parallel programming tool integration. He is on the EPSRC's Technical Opportunities Panel, EPSRC's High Performance Computing: Technology Watch Panel and is also a member of the Technology Applications Sub-Committee of the Joint Information Systems Committee (JISC). He is actively interested in the successful establishment of our Electronic Business Centre and also our Digital Library Research Centre. His personal research interests now extend to explorations of implementations of quantum computing.



Ken Meacham graduated from Southampton University in 1990, with a BSc in Chemistry with Computer Studies, and went on to study for a PhD in molecular modelling in the Chemistry Department. He joined the Parallel Applications Centre in 1994, and helped to develop a Molecular Dynamics (MD) code for Unilever, to study the shearing of bilayers of fabric conditioner molecules. Other chemistry-based work included parallelising the GROMOS87 MD software. Since then, Ken has shifted focus to help develop the PAC's Intrepid package, for intelligent management of meta-application and resources, using performance prediction tech-

niques. He helped develop the PROMENVIR product, for stochastic parametric analysis, and his current project is to develop secure agents to facilitate pay-per-use of remote software and hardware resources.

Juraj Papay graduated from the Electrotechnical Institute in Saint Petersburg, with a Diploma in electronics and Computing. He received a PhD in Computer Science from the University of Warwick in 1997, then joined the Parallel Applications Centre. Since then, Juraj has worked on several ESPRIT projects including, PROMENVIR, HPC-VAO and TOOLSHED. His main interests are in parallel computation, performance evaluation and software design.

Dr. Mike Surridge obtained his PhD in Theoretical Physics in 1986, and moved shortly afterwards into parallel computing research. During the late 1980s he worked mainly on the practical implementation of user-friendly message-passing environments for parallel systems. This work produced one of the first general purpose routing kernels for transputer systems, and interfaces supporting imperative and declarative programming of communicating processes. Mike joined the PAC in 1991, focusing initially on scientific and engineering applications, including the migration to parallel systems of several software packages including MSC/Nastran, PAFEC-FE, PHOENICS (from CHAM), VECTIS (from Ricardo) and CERIUS (from MSI/Biosym), etc. Subsequently, he worked mainly on practical applications of distributed cluster and meta-computing, and on the management of such applications using predictive resourcing models, internet technology and agents. Mike became responsible for Industrial Applications at the PAC in 1995, and was recently appointed as the PAC's Operations Director.