Performance-based Workload Management for Grid Computing

Draft Paper

D.P. Spooner[1⋆], J. Cao, J.D. Turner, S.A. Jarvis, D.N. Dillenberger[2], Subhash Saini[3], and G.R. Nudd

[1] High Performance Systems Group, University of Warwick, UK.
dps@dcs.warwick.ac.uk
[2] IBM TJ Watson Research Center, New York, USA.
engd@us.ibm.com
[3] NASA Ames Research Center, Moffett Field, California, USA.

Abstract. This paper describes a multi-tiered framework (Titan) that applies performance prediction methodologies to the key issue of grid workload management. As grid environments develop to support large-scale computing and data-intensive applications, often distributed over multiple physical administrative domains, it is increasingly important that resources and applications are managed effectively in order to deliver guaranteed service quality to differentiated users. In this work, we demonstrate the improvement obtained when performance prediction techniques are combined with performance-aware middleware components to assist in the process of grid workload management.

Keywords: *Performance prediction, Grid, Workload management, QoS*

1 Introduction

The grid computing paradigm [1, 2], aims to provide a unified worldwide infrastructure for business and scientific computing, through the combination of commodity, specialised and high performance computing systems. A principal aim is to create integrated computational environments that provide technology and resource infrastructures for the efficient use of remote platforms delivering a steady, reliable source of computing power.

There are, however, many challenges when trying to effectively harness the capabilities offered by such systems. The unpredictable and dynamic behaviour of grid resources with respect to connectivity, load and contention create difficulties when attempting to identify suitable resources that can execute an application and deliver guaranteed levels of service. In addition, resource heterogeneity and the loss of central administration effect the characteristics of workload management tools, differentiating this area from conventional multiprocessor and localised cluster scheduling.

⋆ Submitted to 3rd International Workshop on Grid Computing, Baltimore, MD. In conjunction with SuperComputing 2002.

In this work, grid workload management is approached through the use of a *performance prediction* tool to provide accurate estimations of application behaviour prior to run-time. In particular, the PACE [3] (Performance Analysis and Characterisation Environment) system is used to provide performance prediction of both sequential and parallel applications. This performance modelling system separates software, hardware and parallelisation strategies; its capabilities and accuracy having been demonstrated by a number of successful case study applications [4–6]. The layered approach of PACE provides flexibility that can be used to predict the performance of applications on multiple architectures, and the models are typically parameterised so that the scaling effect of additional processors (or distributed nodes) can be investigated.

A key benefit of PACE is the ability to evaluate performance models with differing hardware models *on–the–fly*. This allows a workload management environment to utilise performance prediction when allocating independent tasks to groups of distributed, heterogeneous computing systems. This paper introduces one such system, known as Titan [7], that adopts a multi-tiered approach to workload management, where standard grid protocols and portals are used at the top tier, a collection of application routing brokers are used at the middle tier and localised workload managers are used at the lowest tier.

There are a number of other grid resource management and scheduling solutions that address similar issues including Legion [8], NetSolve [9], Condor [10], Ninf [11], AppLeS [12] and Nimrod/G [13]. While many of these projects utilise query-based mechanisms for resource discovery and advertisement, an agent-based approach is adopted in this work. Nimrod/G estimates the performance of a particular grid resource through heuristics and historical information, while the performance prediction tool PACE is utilised in this context. In the Condor system, scheduling aims to maximise the utilisation of grid resources (resource-orientated); Nimrod and this work provides a system that meets service deadlines (user-orientated).

The organisation of this paper is as follows: section 2 introduces the performance prediction tools in more detail, and how a performance prediction system can assist in grid workload management; in section 3, the system architecture is described along with three methodologies for dispersing tasks to the routing brokers; section 4 presents a case study and demonstrates that the performance models and broker system offer a good improvement over simpler techniques. Conclusions are presented in section 5.

2 Performance Prediction

The predictive capabilities of PACE form the centre point of this workload management framework, providing decision-support for the brokers at the middle tier, and also improving the capabilities of the localised schedulers at the lowest tier. In this section, the PACE system and its associated tools are described.

2.1 The PACE System

PACE models are used to predict the behaviour of a program by means of a set of hierarchical and clearly defined layers. The uppermost layer of a PACE model represents an application as collection of subtasks, where each subtask is a sequential block of work described using control flow and resource usage information. The interaction between these subtasks is specified using a parallelisation template – the second layer of the system which allows the computation-communication interactions between processors to be defined. The lowest layer in a PACE model provides a characterisation of the underlying hardware which is derived from a series of micro–benchmarks.

Combining these layers provides a complex representation of an application which can be used to establish prediction metrics for the execution of the application on a target platform. The level of detail in the model can be chosen to improve accuracy (or reduce the time which it takes to generate the performance estimate), and models can be accumulated and stored in a library for re-use. The use of separate parallelisation strategies also provides a convenient method of analysing the effects that different forms of parallelism have on the resource usage and execution time. The core components of the PACE system include application tools, resource tools and an evaluation engine as illustrated in figure 1.

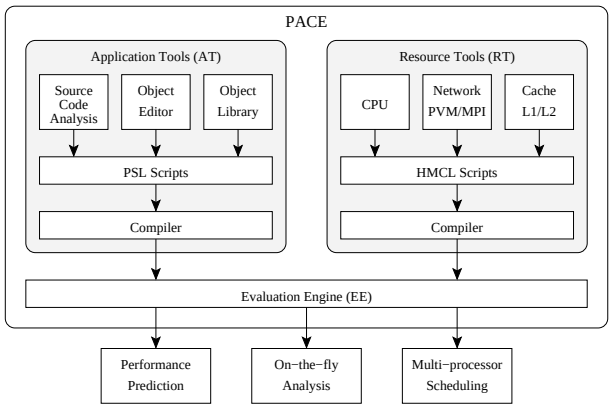


Fig. 1. System breakdown of the PACE tool.

Application Tools

Each application subtask and its parallelisation strategy are described by a performance specification language (PSL). It is the automatic generation of this PSL that comprises the core functionality of the PACE application tools. A source code analyser converts the application code into performance description

definitions and sequential subtask objects. It is possible for the developer to edit these objects to add probabilities to loop and conditional statements which might ultimately improve the accuracy of the prediction (particularly for data- or input-dependent applications). A number of existing objects are stored in the object library and can be re-used if required.

The performance descriptions are subsequently combined into a single application PSL script which is then compiled into an *application model*. This forms one of the inputs into the Evaluation Engine, which itself acts as a repository and analysis tool for the application-level performance information.

Resource Tools

A dedicated hardware modelling and configuration language (HMCL) is used to define the computing environment in terms of its constituent components. The Resource Tools provide several benchmarking programs to measure the performance of CPU, network and memory of alternative hardware platforms. The measurements are represented in HMCL scripts and combined to form *resource models*. This system-level performance information provides a second input into the evaluation engine.

Evaluation Engine

The evaluation engine is the core of the PACE system. It combines the application and resource models to produce results that include performance prediction estimates, detailed trace information of the expected program behaviour and scalability estimates that can be used as the basis for workload management.

2.2 Decision Support for Workload Management

It is the parametric capability of PACE that allows the system to be applied to scheduling and workload management. By evaluating different resource models and varying parameters such as the number of processors, it is possible to evaluate different *performance scenarios* and use the results as decision-making support for workload managers and application routing brokers. The models are typically evaluated in seconds and as new resources are introduced into the grid, only a new hardware layer is required - the defined separation in the PACE layers permits re-use of the application subtask and parallelisation definitions.

3 Grid Workload Management

As a standalone performance prediction and evaluation tool, PACE can be used to tune and optimise application execution behaviour. In this work, the system is used as key component in a hierarchical structure of application routing brokers, localised workload managers and portal/interoperability providers. While all of these components may reside on the same system, they represent different tiers as illustrated in figure 2 and have different goals.

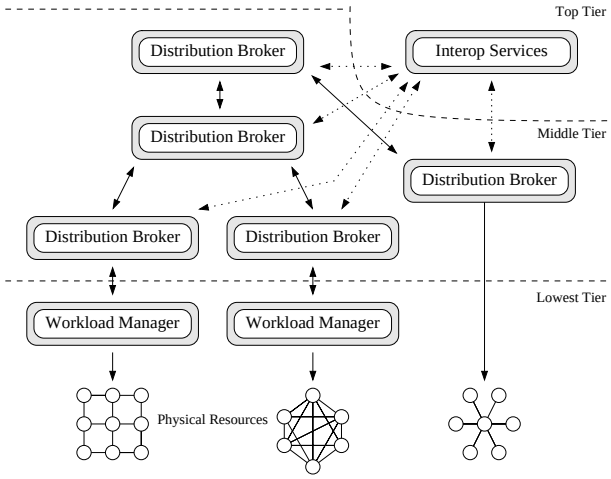


Fig. 2. The Titan architecture consisting of application distribution brokers that communicate with other brokers and workload managers to execute tasks in accordance with service policy constraints. An interoperability interface allows task submissions to particular brokers.

3.1 Titan Framework

The top-tier provides an interoperability layer that allows tasks to be submitted to a suitable application routing broker. This tier represents the entry point for the user and is currently implemented as a simple portal, although work on integrating with Globus is currently in progress.

The middle-tier encompasses the brokers that utilise the PACE system to make routing decisions based on predicted execution time and user-specified service deadlines. The brokers adopt an agent-based methodology [14], which has been developed for the management of large-scale distributed systems with highly dynamic behaviour. They are homogeneous in their capabilities (although configuration options will influence behaviour) and are arranged into a hierarchy organised by federated groups. Each broker can be considered both a service provider and requester, and all have capabilities for advertising and discovering resource data to the users and each other. The hierarchical structure is used to exchange capability information (such as schedule makespans and the resource architectures that the brokers represent).

Internally, the brokers are composed of a communication layer that performs functions that interface to the external environment and include the ability to receive service advertisement and discovery messages from other brokers, and a coordination layer which contains four key components: a capability table (CT) manager which maintains a cache of discovery and advertisement details, a PACE evaluation engine, connectivity to the local workload manager, and a

matchmaker. These components work together to make decisions as to how a broker should act when messages arrive from the communication layer.

The lowest-tier consists of a workload manager local to the broker that queues tasks on a local set of resources. It uses the performance models with an iterative heuristic algorithm to try different scaling combinations so that tasks can be packed tightly, reducing the end to end makespan and processor idle time. To a lesser degree, service deadlines are also considered at this level. This algorithm creates multiple schedules based on the current task queue, and uses an objective function to select successful schedules. These are subsequently combined using appropriate operators and the process is repeated. The objective function is derived from the makespan of the current schedule which can be predicted using PACE for the given number of processors and a resource type. The function also accounts for the amount of idle time in the schedule (where processors are blocked or unused), and the number of service deadlines that are met.

3.2 Workload Flow

On receipt of an application, the broker typically interrogates the local workload manager by means of the adaptor in the coordination layer to ascertain whether the task can execute on the locally available resources and meet user-specified service quality metrics (QoS contracts). If successful, the tasks is submitted to the local manager for scheduling. If not, the broker will attempt to locate a remote resource using either knowledge of other brokers found through periodic advertisement, or will initiate a discovery process that will defer the task to a broker further up the hierarchy.

When a broker receives a deferred request, it will act out the same processes as the original broker. Being further up the tree, it will usually have larger capability tables and hence more details of available resources. However, it is possible that the task will eventually reach the root broker and knows of no lower broker that can meet the user requirements. The discovery process is terminated, and the task returned to the broker that could execute the task in the earliest time with an instruction that deferral is not permitted.

4 Case Study

In this section a case study is described that demonstrates the operation of the Titan workload management system compared with two other task dispersion methods. The experiments involve the distribution of a series of independent parallel tasks, selected from a group of scientific codes including an FFT code, a bitonic memory sort and the ASCI Sweep3D parallel benchmark [4]. The service deadlines (the time required to complete the application) are randomly selected for each application from a range of values derived from the predicted run times.

4.1 Experimental Design

The objective of the experiment is twofold: firstly, to contrast the characteristics of using three methods of workload management; and secondly: to observe the scaling effects of these systems under different loads and with an increasingly large experimental grid.

In the first set of experiments, the system is tested with four resource brokers. In each subsequent experiment the number of brokers is increased by two, to a maximum of twenty. Each broker represents a cluster of 16 homogeneous processor nodes, which may be sixteen 16 separate uniprocessor machines or a single 16-node multi processor system. Figure 3 illustrates the arrangement of all twenty brokers (the early experiments use a subset of this graph), and lists the architecture that each broker represents (from a set of Sun Sparcs 2/5, Sun Ultra 1/5/10 and SGI Origin 2000s). This provides an experimental grid with an effective process capability of 64 nodes (with four brokers) to 320 nodes (with twenty brokers). The hierarchical arrangement was selected to represent a virtual organisation with four sub organisations (B-02 to B-05) with contributory resources.

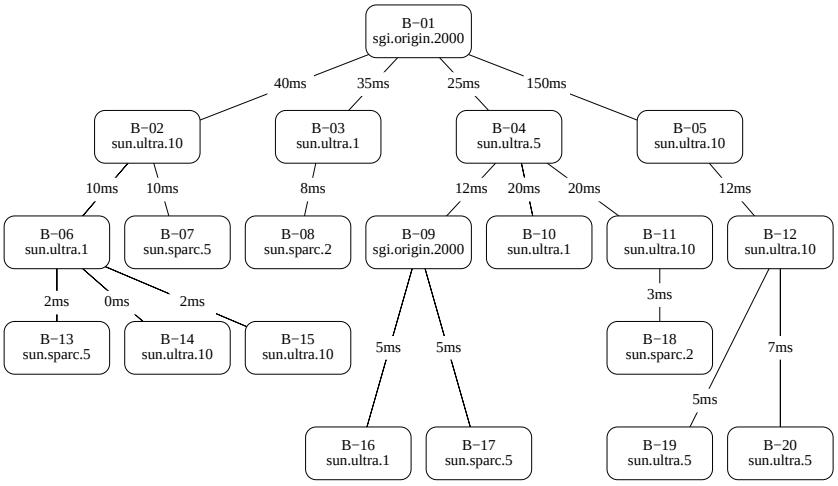


Fig. 3. Topology of the broker structure with the architecture types that the brokers represent. The communication delays between the brokers are labelled on the node links.

For the purposes of this experiment, each broker maintains a matrix of timing delays which are imposed on the messages that arrives from other brokers. Although this has no effect on the available bandwidth between brokers, it does effect the message latency, which goes some way toward simulating the effect of running these experiments over a wide-area network infrastructure.

Three levels of workload are created for the purpose of these experiments, including: *light* which equates to one task every two seconds, *medium* which equates to one task every second, and *heavy* which equates to two tasks every second.

Dispersion Methods

Three workload management strategies are used to disperse and execute the tasks on the experimental grids. The first two strategies 'spray' the tasks onto the brokers directly with an instruction that prevents deferral to other brokers. In effect, the hierarchical structure is removed and brokers can only run tasks on their local workload managers. These strategies are described below and the results are compared in the next section.

The first strategy (*Spray QLen*) is a simple technique that does not use performance models to steer tasks to suitable architectures, and does not use any optimising strategy to reduce the makespan on the local workload manager. This dispersion strategy requires each broker to reply to a request from the initiating broker for their 'queue length' – this being the sum of tasks in the executing and to-be-scheduled state. This provides an indication of how busy each component cluster of the grid is and can steer the scheduling decision accordingly. When all of the resource brokers have responded, the system with the lowest queue is selected and the task allocated to that system.

While this technique works well with small numbers of entirely (or near) homogeneous systems, it performs less well with diverse architectures as it takes no account of the underlying hardware architecture and will treat a slow resource with a small queue preferentially to a fast resource with a larger queue. This spraying technique is more suited to server farms and closed environments rather than large wide-area infrastructures where scalability issues can become a problem. However, it should be noted that the implementation described here represents the simplest spraying technique and this method could be improved by using a resource hardware scalar that weights the queue lengths depending on the resource type.

The second workload management strategy (*Spray Perf*) also requires a response from each broker. In this case, a request is made for the predicted makespan which is the expected time that a new job would begin execution on the resources the broker represents. The system can then predict the expected completion time for the particular application (predicted makespan plus predicted execution time), and the task allocated to the broker that can complete the task at the earliest time.

The third workload management strategy (*Broker*) is the native technique provided by the Titan system with the hierarchical relationships reestablished. The application routing brokers can manage the distribution of tasks based on predicted execution results, referring tasks where necessary. On receipt of a task, a local broker can decide to submit the task to the local workload manager if the deadline can met. Alternatively, it may use its knowledge of other brokers (through CT exchange) to direct the task to a more appropriate (and perhaps

less-loaded) architecture. Finally, it may initiate a discovery process to locate suitable resources – this can be viewed as a managed version of the performance spraying technique described previously. The advantage of this system is that a task submission does not necessarily initiate a sweep of the entire broker hierarchy as relevant data may already be available, and hence the communication overhead and service requirements are more evenly balanced

4.2 Evaluation

Three metrics are considered in the comparison of these dispersion techniques. The first metric is makespan which is a measure of the end-to-end time of the scheduler. This represents the difference between the start time of the first job and the completion time of the last job. In the analysis presented below, makespan refers to the average makespan of all the brokers in the experimental grid.

The second metric measures the time difference between the completion time of a particular application and the user-specified requirement. This is then averaged for all the applications and brokers. Negative values indicate that most tasks are failing their deadline. In this experiment, this is expected in the early cases as there are only a few brokers (and hence fewer processing nodes) to run the tasks.

Finally, the number of packets emissions are measured at each broker to obtain an indication of the communication overhead. This is a key metric when dealing with grids that may expand across many organisational and regional boundaries.

Results

As the workload increases (from 1 job per second to 2 jobs per second), the makespan increases accordingly. It is also seen that when more grid resources are added, the makespan correspondingly decreases. Using the *Queue* method, only very coarse-grained workload schedules can be achieved and the resulting makespan is worse than those of the prediction-based *Spray* and *Broker* methods. The difference is more obvious when the number of grid resources are small and the system workload is relatively high, this is because in these situations resources are critical and inappropriate allocations made by the queueing method are cumulative. The results of the *Spray* and *Broker* techniques are also compared. As previously described, *Spray* aims to provide a global optimisation and the *Broker* technique offers a more network-managed solution by only communicating with neighbouring brokers. It is therefore reasonable that in most situations, *Spray* can achieve a better makespan. However, it can be observed that the results of the *Broker* technique are comparable to those of *Spray*, which is a good trade-off considering the network overheads described below.

As the workload increases, the capability of the system to meet the deadlines is decreased. Similarly, when the workload is fixed and more grid resources are included, the more chance the system has of meeting the collective deadlines. The

results of deadline management are similar to those of makespan. The results of using the *Queue* method are much worse than the other two techniques and the *Spray* method results in a slight improvement in the maintenance of deadline.

The number of network messages used for resource advertisement and discovery increases linearly with the number of brokers and number of and jobs. Using the *Queue* and *Spray* methods means that there are no messages for resource advertisement. However, the *Broker* technique uses the concept of resource advertisement and discovery to significantly decrease the amount of network traffic. It is clear that the mechanism of only passing messages among neighbouring brokers improves the scalability of the system as the number of brokers increases.

In summary, using techniques which employ performance prediction can significantly improve the scheduling and demonstrate an advantage on resource utilisation and application executions, especially when system workload is high and grid resources are critical. And the use of a distributed broker mechanism can reduce the network overhead significantly and make the system scale well rather than using an centralised control, as well as achieving a reasonable good resource utilisation and meeting application execution deadlines.

5 Conclusions and Future Work

The work presented in this paper is concerned with improving workload management in grid environments using a multi-tiered framework based on performance prediction. The system architecture is described, along with case study results that compare Titan with a simple spraying technique and a more advanced spraying algorithm based on PACE. The results support the argument that where performance models exist, makespan and deadline metrics can be improved on heterogeneous grid infrastructures using predictive information.

Furthermore, the use of the performance evaluation engine as a decision support system for the distributed application routing brokers clearly lead to a reduction in communication costs, improving scalability over a request orientated technique.

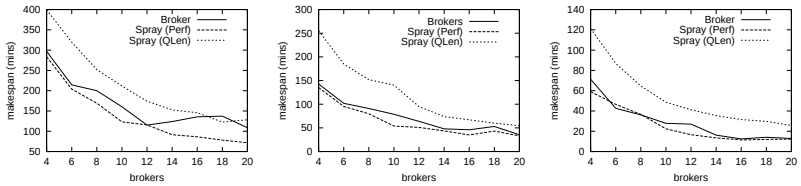
Future work will concentrate on providing higher-level services to manage application work flows, and adding the capabilities for the brokers to tune their performance dynamically. It is also planned to integrate Titan as a Globus service that can provide effective workload management services for situations where performance models are available.

Acknowledgements

This work is sponsored in part by grants from the NASA AMES Research Center (administrated by USARDSG, contract no. N68171-01-C-9012) and the EPSRC (contract no. GR/R47424/01).

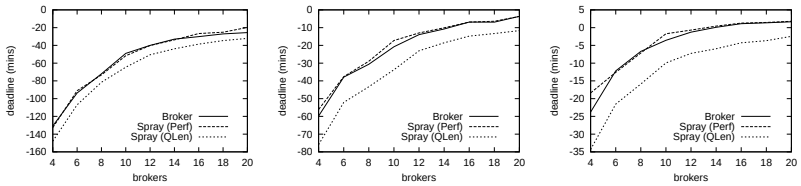
References

1. Foster, I., Kesselman, C.: The GRID: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc. (1999)
2. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organisations. Intl J. of Supercomputing Applications (2001)
3. Nudd, G., Kerbyson, D., Papaefstathiou, E., Perry, S., Harper, J., Wilcox, D.: Pace : A toolset for the performance prediction of parallel and distributed systems. Int. J. of High Performance Computing Applications, Special Issues on Performance Modelling **14(3)** (2000) 228–251
4. Cao, J., Kerbyson, D., Papaefstathiou, E., Nudd, G.: Performance modeling of parallel and distributed computing using pace, Phoenix, USA (2000) 97–104
5. Alkindi, A., Kerbyson, D., Papaefstathiou, E., Nudd, G.: Optimisation of application execution on dynamic systems. Future Generation Computer Systems **17(8)** (2001) 941–949
6. Turner, J., Lopez-Hernandez, R., Kerbyson, D., Nudd, G.: Performance optimisation of a lossless compression algorithm using the pace toolkit. Research Report 389 (2002)
7. Spooner, D., Cao, J., Turner, J., Keung, H.L.C., Jarvis, S., Nudd, G.: Localised workload management using performance prediction and qos contracts. 18th Annual UK Performance Engineering Workshop (2002)
8. Chapin, S., Katramatos, D., Karpovich, J., Grimshaw, A.: Resource management in legion. Future Generation Computer Systems **15(5)** (1999) 583–594
9. Casanova, H., Dongarra, J.: Applying netsolve's network-enabled server. IEEE Computational Science and Engineering **5** (1998) 57–67
10. Raman, R., Livny, M., Solomon, M.: Matchmaking: Distributed resource management for high throughput computing. Proc. of 7th IEEE Int. Symp. on High Performance Distributed Computing (1998)
11. Takefusa, A., Matsuoka, S., Nakada, H., Aida, K., Nagashima, U.: Overview of a performance evaluation system for global computing scheduling algorithms. Proc. of 8th IEEE Int. Symp. on High Performance Distributed Computing (1999) 97–104
12. Berman, F., Wolski, R., Figueira, S., Schopf, J., Shao, G.: Application-level scheduling on distributed heterogeneous networks. Proc. of Supercomputing (1996.)
13. Buyya, R., Abramson, D., Giddy, J.: Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. Proc. of 4th Int. Conf. on High Performance Computing (2000)
14. Cao, J., Kerbyson, D., Nudd, G.: High performance service discovery in large-scale multi-agent and mobile-agent systems. Int. J. of Software Engineering and Knowledge Engineering, Special Issue on Multi-Agent Systems and Mobile Agents **11(5)** (2001) 621–641



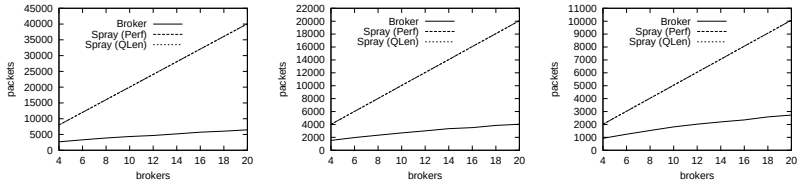
(a) (b) (c)

Fig. 4. Comparison of makespan measurements for broker, spray (perf) and spray (queue length) dispersion methods. Graphs (a), (b) and (c) demonstrate the behaviour under workloads of 2 tasks/s, 1 task/s and 1 task/2s.



(a) (b) (c)

Fig. 5. Comparison of makespan measurements for broker, spray (perf) and spray (queue length) dispersion methods. Deadline refers to the number of seconds in advance of the user requirements. As the deadline times in this experiment are short and there are initially few brokers, this is generally negative (i.e. deadline is missed on average). As before, graphs (a), (b) and (c) represent the same workload conditions as the makespan figures.



(a) (b) (c)

Fig. 6. Comparison of packet emissions for broker, spray (perf) and spray (queue length) dispersion methods. In these diagrams the the spraying lines have occluded each other as their query-based mechanisms result in a linear increase in packets. In comparison, the brokers minimise the communication overhead as many messages are consolidated into broker 'adverts'.