Simulating a Computational Grid with Networked Animat Agents

K. A. Hawick and H. A. James

2006

Computational grids are now widespread, but their large-scale behaviour is still poorly understood. We report on some calculations of loading, scaling and utilisation behaviours of computational grids, based on simulations. We employ animat agents on a topologically detailed graph representing a grid overlay network. Agents are used to represent computational jobs and users. We can obtain realistic behaviours by endowing user agents with time-varying microscopic behaviour patterns. We are able to study the static flow and dynamical macroscopic properties of the network including emergent pathological behaviours and other anomalies that arise when parts of the network become temporarily available. Our model is based on graph theory with various attributes decorating the edges and nodes which have physical locations. We develop some overall grid utility metrics that can be analysed. Our work suggests that grids do need to be treated as complex adaptive systems.

Keywords: networked agents; network graph; spatial complexity; Monte Carlo simulation

Simulating a Computational Grid with Networked Animat Agents

K.A. Hawick   **H.A. James**

Institute of Information and Mathematical Sciences
Massey University – Albany
North Shore 102-904, Auckland, New Zealand
Email: {k.a.hawick,h.a.james}@massey.ac.nz
Tel: +64 9 414 0800   Fax: +64 9 441 8181

Abstract

Computational grids are now widespread, but their large-scale behaviour is still poorly understood. We report on some calculations of loading, scaling and utilisation behaviours of computational grids, based on simulations. We employ animat agents on a topologically detailed graph representing a grid overlay network. Agents are used to represent computational jobs, users and resources. We can obtain realistic behaviours by endowing user agents with time-varying microscopic behaviour patterns. We are able to study the static flow and dynamical macroscopic properties of the network including emergent pathological behaviours and other anomalies that arise when parts of the network become temporarily unavailable. Our model is based on graph theory with various attributes decorating the edges and nodes which have physical locations. We develop some overall grid utility metrics that can be analysed. Our work suggests that grids do need to be treated as complex adaptive systems.

Keywords: networked agents; network graph; spatial complexity; Monte Carlo simulation.

1   Introduction

Computational grids(Foster and Kesselman 2003) are complex systems involving many separately autonomous users and computational resource elements. The large-scale behaviour of modern grids is still poorly understood, but is non-trivial to simulate effectively. We model some computational grid phenomena using an agent-based simulation model, where the grid overlay on the Internet is described by a graph model.

The main purpose of a grid simulation is to develop an understanding of the overall system behaviour under various user and resource conditions. It is useful to investigate "hot-spots" in the grid network itself, and the flow of grid jobs from users to matching resources. There are other "what-if" questions that can be posed with a high-level simulation, namely:

- where to locate resources to best effect?;

- are existing resources being used to best effect?;

- what will be the effects of various management or scheduling policies?;

- are we obtaining good value for money with the placement of existing resources; and,

- what local decisions are affected by grid-wide decisions?

Ultimately the question of greatest importance for a simulation is to test the fundamental grid hypothesis – is it in fact worthwhile either for the users or resource owners or both, to have the resources be part of a grid?

Other researchers have investigated these important questions for grid systems. Most other work has used a packet based formulation, and consequently the simulated grid is really an emergent set of behaviours from a network simulation. We take a higher-level approach and model the grid as an interacting set of agents: static users and resource agents and mobile job agents. We perform a Monte Carlo simulation of a large set of interacting microscopic agents to observe the emergent grid system phenomena at a macroscopic level of measurement. One important feature of our prototype system is that we employ a detailed and physically embodied network model that is described by a graph connecting specific hosts and grid resources. Agents decorate the nodes and edges of the graph.

In this paper we present some preliminary work on our agent-based grid simulation. In section 2 we discuss some important ideas from various software simulation systems, and describe our agent simulation framework. We present some test grid network patterns in section 3 and show how they can be used to provide an environment for the detailed agents that we describe in section 4. We show some selected results and grid phenomena in section 5 and discuss some of our assumptions for the model in section 6, along with some ideas for future development of agent-based grid models.

2   Simulation Software Infrastructure

A number of authors have described simulation environments suitable for models of grids and networks (Sulistio, *et al* 2005). Some systems are based on the use of threads which, while useful for attaining concurrency, can constrain the freedom of a simulation to impose its own scheduling and concurrency behaviours and policies(Robertson, *et al* 1993, Ibbett, *et al* 1996). An excellent review of simulations systems is given in (Sulistio, *et al* 2004). We needed to develop a new framework to support the embedding of a network graph with spatial attributes. Our network models can be therefore be rendered for visualisation as well as used as the substrate for calculations.
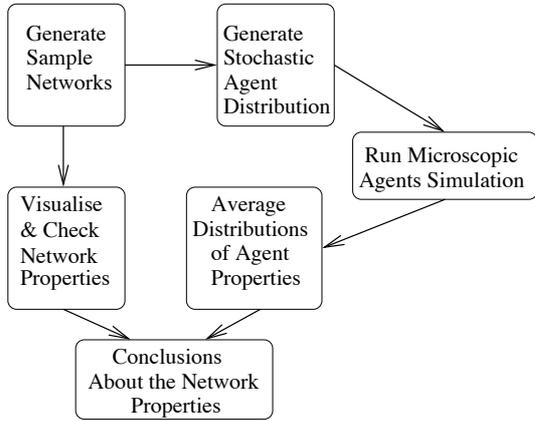
Figure 1: Our experiment methodology - grid graph networks are both visualised and used as the substrate for microscopic agent calculations, from which agent-behaviour distributions can be measured and properties of the network as a whole inferred.

Figure 1 indicates the methodology behind pour experiments and our need for a simulation framework that supports separate and visualisable spatial graphs to implement the network substrates.

Our system is based on an underpinning directed graph data structure, where nodes are wired up using FIFO queues and an encapsulating data type (known as a Thaum[1]) is used to ferry mobile simulation agents around the network. Static agents can decorate the nodes of the network and can read their queued inputs and write their outputs to queues. It is of course possible to place size constraints on the queues. Research results relating to channel based systems like Occam (Roscoe and Hoare 1988, Hoare 1985) and network models such as Petri nets (Girault and Valk 2003) pertain to the effects of queue sizes and the blocking behaviours that can ensue. In the work we describe here, our mobile agents are "grid jobs" and are assumed to be relatively compact in size and can therefore be easily queued. We monitor queue sizes but do not constrain them.

The agents in are system are simulated using a stochastic Monte Carlo procedure. Individual agent behaviours are deterministic but a stream of random numbers is used to specify starting conditions and also the order of agent firings. Since averaged lifetimes of agents converge to stable values when we release large numbers of agents on the same network, it seems we are randomly sampling and hence measuring properties of the network, for that particular sort of agent behaviour.

The agents in our model are given a fair but randomly ordered opportunity to fire or carry out their "program" each time step. The networked agent model therefore schedules agents fairly but avoids sweeping or correlation effects that would arise from firing agents in an unchanging order. We employ a simple permutation vector to remove sweeping effects. Individual agent programs can be configured to read and store available inputs while they await unavailable input. For simplicity however we assume that our agents in the model described here only proceed when all needed inputs are available, and that any storing or partial inputs is done by the arc queues. By allowing unlimited output queue sizes, we also allow agent programs to write immediately to their out-

---

[1]For those who need to know, a "Thaum" is the "unit of magic" in Pratchett's Discworld (Pratchett 2000)

puts without the need to block. These non-blocking assumptions are useful in simplifying the interpretation of simulation time for our model. We treat time as fairly coarse grained with time units more related to the application model than to artifacts of the network queueing infrastructure implementation.

We implemented the model using both the Standard Template Library Queue data structures in C++ (Hewlett-Packard Company 1994) and also the generic Queue in the Java Utilities package for Java Development Kit (JDK) 1.5 (Sun Microsystems 2005)). It is also possible to use a hand written non-generic queue container (implemented around an existing library container such as a linked-list for instance). We employed the latter approach in implementing the self simulation features of the original DISCWorld "pre-grid" metacomputing system (Hawick, *et al* 1989). For the purposes of simulation either the C++/STL or the JDK1.5 `Queue<Thaum>` approach is entirely sufficient for modelling efficiency.

We treat the network model as a directed graph and we use separate queues for the two different directions of agent flow. This makes sense in practice, although we do need to keep track of the cumulative activity on both queues that reside on an edge in the network, in order to make sensible load average measurements.
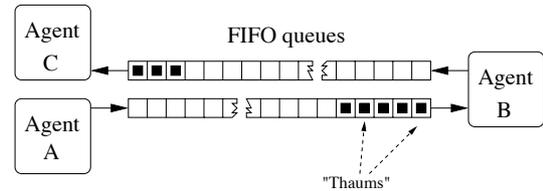

Figure 2: Three agents communicating via FIFO queues of Thaum tokens. "A" is an injector, "B" is a filter, and "C" is a consumer of Thaum-jobs. Thaums encapsulate mobile job "agents".

Figure 2 illustrates the basic network apparatus. Nodes A, B and C will have static agent programs residing on them, and they can manage the flow of mobile agents along the interconnecting FIFO queues using a simple push/pop/test interface. Individual static agents can take the form of injectors of mobile agent jobs (ie users); consumers (ie gridded resources that satisfy job requests); and filters that act as spine points in the network (ie resource discovery agents.) Individual agent models can be written to extend one of these basic patterns.

The advantages of this graph-based network infrastructure are that agents can be written using a small set of interface calls to use local information only. They can be managed by a set of infra-structural policies for implementing "simulation time" and scheduling fairness that is global but effectively opaque to individual agents. Statistics on the overall system behaviour can be collected independently of agent code since the apparatus is built into the network containers for nodes and edges and into the Thaum object infrastructure itself.

The architecture of our model is shown in figure 3. A `ThaumQueue` has been created to properly represent a potentially-unbounded queue of Thaums. This data type also allows us to maintain statistics on the number of Thaums currently in the system and their relative ages when they are consumed by resources. A basic Machine is defined to provide the minimum functionality of a network entity. This functionality is extended by the Graphical Machine
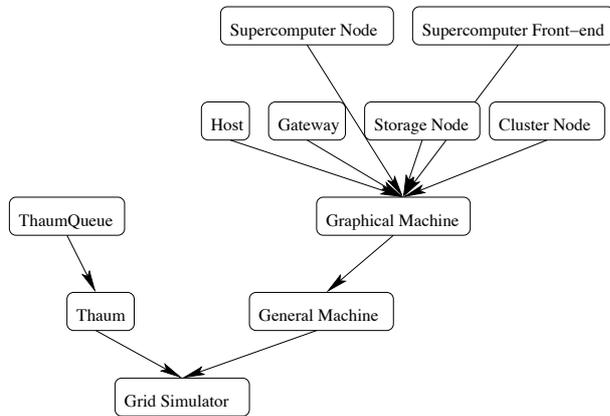
Figure 3: Object architecture of our model. A basic machine specification is provided that contains the functionality to 'fire' agent actions for those Thaums on input queues. The concept of 'firing' is then made specific to the type of resource each machine represents. A Graphical Machine has been layered to provide the functionality to perform graph analysis on the simulation model.

class, which provides the ability for each network entity to be represented in a 3-D viewing tool such as GraViz (Hawick 2004); this is useful for providing static measurements of the network characteristics and can in fact be used to make animations of network activity.

As discussed in section 4, Machines are further specialised into six different types, each representing a network entity type found on our simulated Grid. This object specialisation means we still have the basic functionality inherited from parent classes, but are able to customise their 'firing' functions.

This model is very close to that of our DISCWorld metacomputing system prototypes and still provides what we believe to be a clean architecture for a grid system that maintains as much information as possible in localised form.

3   Some Test Grid Networks

Using the graph based apparatus described above, we can configure a particular grid graph with users, compute resources and resource discovery agents. Our simulation code uses graphs generated by a graph model editing tool (Hawick 2004) which supports coloured nodes and bi-directional arcs embedded in a three dimensional space. In practice most grid overlay networks are usually planar graphs and we employ grid networks like the one shown in figure 4 to explore the properties of our simulation model.

Figure 4 shows a typical national grid for a small country, with a mix of computational and storage and networking resources and with some international gateway points to the national spine network. In the present paper we do not make full use of all the possible static agent types and just focus on "monochromatic" users, resources and discovery agents, but it is possible to have many different "coloured" sorts of job agents with different properties.

It is possible to generate arbitrary pseudo grid-overlay networks by specifying statistical parameters and numbers of different sorts of nodes. Most actual grids that we know of have a hierarchical structure that is based on hubs and concentrations of high performance resources around them. Grids of high performance resources therefore generally reflect the

physical Internet structure upon which they are overlaid.

It is an interesting problem to investigate what typical parameters are needed to describe realistic grid overlays. For the purposes of this paper however we investigate scaling by simply aggregating together copies of the network in figure 4. Our graph editing tool supports the merging together of sub-networks by importing and combining node points. We set up the unit base model in figure 4 to have "gateway" nodes. We can therefore replicate and paste together multiple instances of the unit network at the gateway points to make arbitrarily large aggregate networks. This is not an unrealistic assumption since many national grids are linked to the international community this way.

This scaled grid of replicated units approach is particularly interesting for an investigation of the fundamental grid hypothesis - that it is indeed worthwhile to gridify computational resources to obtain better user service responses and/or better resource utilisation.

Figure 4 shows the structure of our initial test model. For historical reasons we have called this "model 2". Models 3 and 4 were derived from model 2. Their characteristics are shown in table 3.

For each of the models, damage to the system was simulated by removing some of the arcs or vertices – simulating network down-time or the failure of a network entity. Results from this are discussed in section 5.

4   Agent Model Details

We have described grid network overlay above, but the emergent properties also depend upon the microscopic behaviour details of the user, resource and discovery static agents on the nodes of the network. Agents are encapsulated as Thaums and traverse the (approximately) fixed network that constitutes their world. The world is modelled as a set of resources of different agent types. Agents are given the opportunity to 'fire' once each time step: we have designed the model so that the order in which agents fire is randomised so as to avoid any sweeping update effects but at the same time ensure each experiment is reproducible.

We have implemented the following agent types (colours): users; supercomputer nodes and their front-ends, cluster nodes, and storage nodes. As previously stated, in our simple model, user jobs are monochromatic – they all issue requests for either processing resources or storage resources once per unit time-step: our system does not make a distinction between these two types. A Thaum is created to encapsulate the details of the request, which is then routed through the grid structure until it meets a resource with available capacity to match the request. When a request meets a resource that can satisfy its request, the request is consumed and we say that the processing or storage job has been effectively completed. Thus, the types of jobs that we are primarily considering in this network simulation are simple, mono-processor jobs (future work will incorporate the processing of multi-processor jobs, and jobs consisting of a non-trivial *graph* of tasks). For convenience we make the generalising assumption that on average the users of each host generate one grid job per hour over the whole simulation timeframe. Thus looking at 1 week of simulation time means that on average each host generates $24 \times 7 = 168$ request Thaums, which must be matched by available resources.

In the most common case, users are not really concerned where their jobs are executed, as long as they
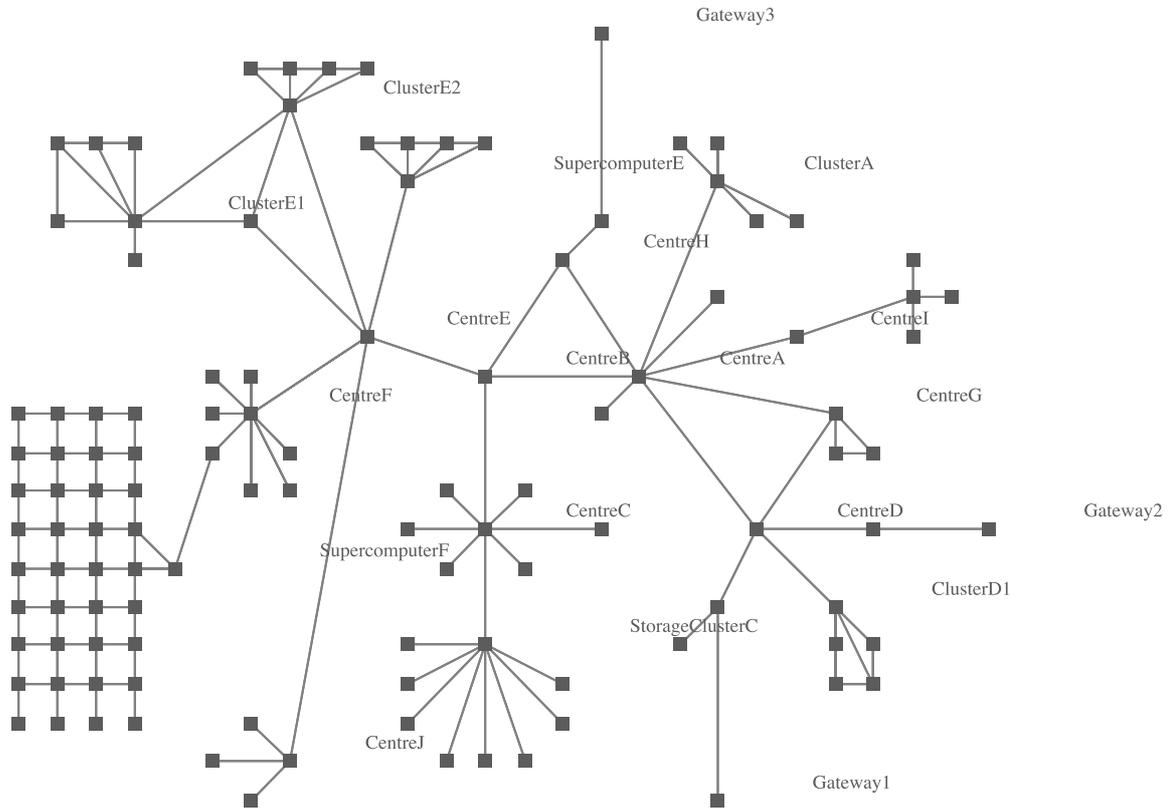
Figure 4: A typical network on Computational Gridded Resources, spatially separate, with heterogeneous resources.

Model	Vertices	Arcs	Users	Supercomputer Nodes	Cluster Nodes	Storage Nodes
2	114	304	28	44	9	14
3	228	610	56	88	18	28
4	456	1224	112	176	36	56

are executed in a reasonable time-frame. To achieve the illusion of this process, we allow the filtering nodes of our graph to instantaneously take all Thaums on their input queues and send them to an output queue. Note that, because in our model Thaums are not sent to any particular destination, there are no routing tables, *per se* – Thaums are routed to output queues using a random throw of the dice in the hope that sending a Thaum down one grid link will lead it to be processed faster than if we sent many down the same link. Future work will model resource discovery and routing more accurately.

In our simulation there are two types of network entities which can inject jobs into the Grid: hosts and gateways. Hosts in our simulation only produce compute requests. They perform no processing of their own, as if they are relatively low-powered desktop machines compared with Grid cluster nodes and supercomputer resources. Whenever a request Thaum is sent to a Host it merely acts as a filter, passing on the Thaum onto one of is output queues. Gateways represent a one-way network filter (similar to the operation of a diode): they allow jobs to travel in one direction only and are meant to represent jobs arriving into this Grid from another network. If a Thaum is sent *to* the Gateway it merely reflects the Thaum onto one of its available output queues.

There are several types of network entities that only represent filters: Spines and Supercomputer Front-ends, and others that fulfil filtering functions as a side-effect of their main function, such as Hosts. Filters merely serve to move Thaums between seg-ments of the network. In the case of a Supercomputer Front-end it is connected to an array of Supercomputer Nodes. We have made the conscious decision to arrange the Nodes as a grid – the effects of this decision are discussed in section 5.

Finally, there are several resources that simply act as sinks for Thaums: Cluster Nodes, Supercomputer nodes, and Storage nodes. The physical parallel of these network entities are entirely obvious, although we remind the reader that in the present model requests are all of the same type: we consider a request matched whether it arrives at either a processing-type node or a storage node. The difference between these types of nodes will become important when we introduce different flavours of Thaum requests. We also make the simplifying assumption that, unlike the real world, part of the time that a Thaum takes to arrive at a processing or storage resource could be taken as an analogue of the time that the request might take to execute in the real world: in our model a Thaum can be consumed once every time step, signifying that a new job can be started each time step. We explicitly do not incorporate into this model a concept of different length processing requests, processing requests that require multiple resources, or large storage requests that would either tie up the storage facilities for a large amount of time or consume a large amount of interconnection bandwidth.

Agents internal states are managed as private data structures with public accessor functions. Each mobile agent carries around information on its current lifetime and potentially on the path it has taken to
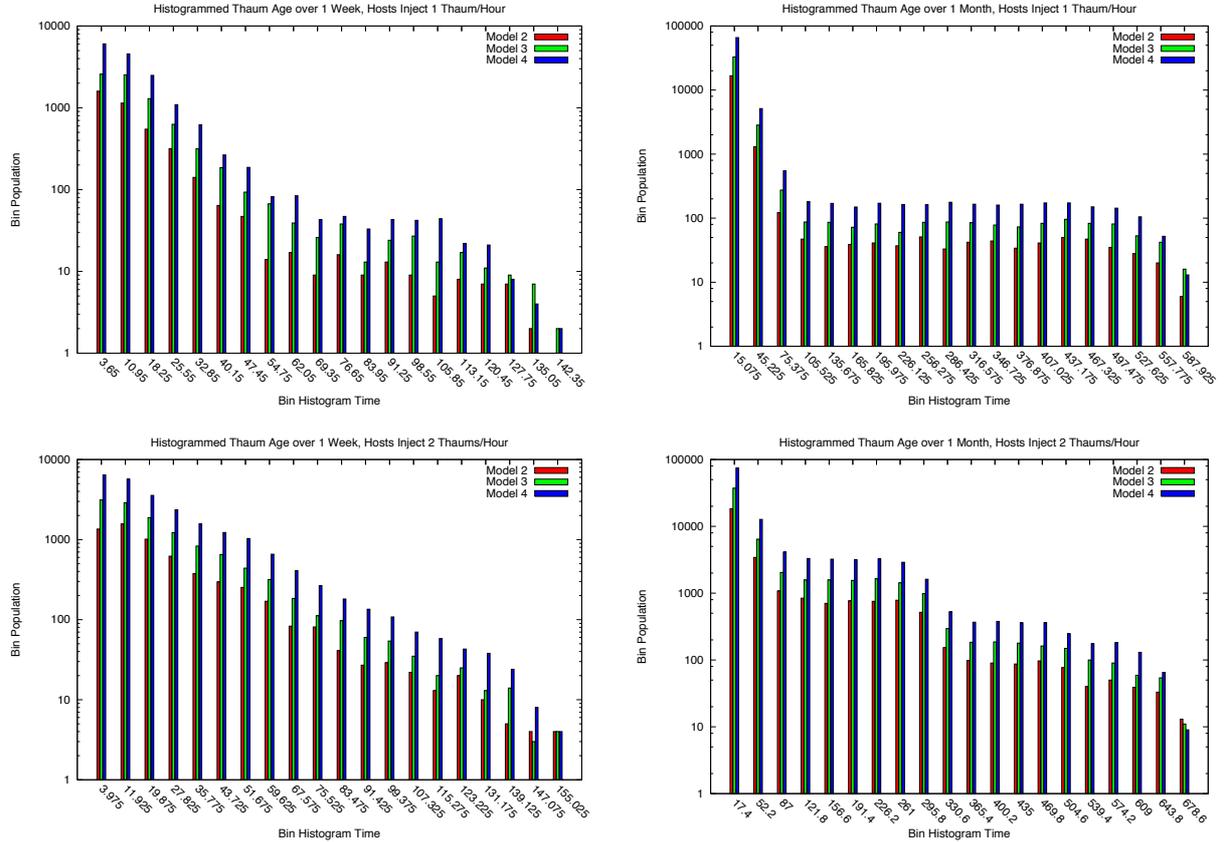
Figure 5: Histogram of Thaum ages when they reach their destination for simulation time of 1 week (left column) and 1 month (right column) for hosts that inject either 1 Thaum per hour (top row) or 2 Thaums per hour (bottom row).

various resources. Our simulation framework includes mechanisms to log information on all agents when they are consumed (i.e. when the job they represent has been satisfied). It is the distributions of the bulk properties of these mobile agents that we study in this paper.

In this paper we focus on just the average time that it takes for a Thaum to find a matching (and available) resource and also the overall resource utilisations across each period of simulation. The model is loosely synchronous in that we model time as a discrete 1 hour global clock, although we allow stochastic actions to occur during each time slot.

The job agents are mobile and are processed by the static agents that live on the nodes of the network model. In future models we hope to investigate the effects of allowing compute and storage resources to send out 'invitations' for their use to Hosts and reservation effects. We also intend to investigate the use of Thaums 'coloured' according to the resources they require.

5 Selected Results and Behaviour Patterns

Figure 4 shows histograms of Thaum ages when they are consumed by the various entities in the network. In each case shown in this figure, the most prevalent time that a Thaum takes to reach an available network resource is less than 20 time steps and for the model in which each host injects a single request Thaum per unit time most requests are satisfied within four time units. This shows that users connected to Hosts local to the larger processing resources such as groups of Cluster processing nodes and also Supercomputer Nodes are able to gain ready

access to those resources, while those users who are perhaps further away from the centres will have to wait a little longer for their requests to be satisfied. Also note that the histograms are drawn using a log-scale for the bin populations. Only a very small number of Thaums are in the network for any longer than about 120 time steps. Note this is a large time compared to the network size – due to Thaum diffusion rather than accurate forward discovery or routing.

Each entity records the number of Thaums that it produces, consumes or filters so that we can visually see the 'hot-spots' in the network. These are shown in figure 6 for model 4, where, predictably, the Spines are the most heavily-loaded Grid feature, followed by a fairly even load across the Cluster Nodes. Of particular interest is the load-information of the model that represents Supercomputer Nodes. The grid structure of the Supercomputer nodes, shown in figure 6 is based around a single Front-end filter node that is connected to the nearby Supercomputer Nodes. Each Node is connected to the neighbouring nodes, but not necessarily a constant number of nodes, not necessarily to the Front-end. Thus, in this figure we can see that the nodes close to the Front-end are quite heavily utilised, but those further away are comparatively under-utilised. This is in contrast to the much smoother load graph shown in figure 7, in which the number of Thaums injected into the Grid per unit time has been doubled.

It is interesting to note that removing a major link between the connected quadrants of model 4 does not increase the average time that a Thaum takes to find an appropriate resource, however it does cause the traffic through other related Spines to nearly double.

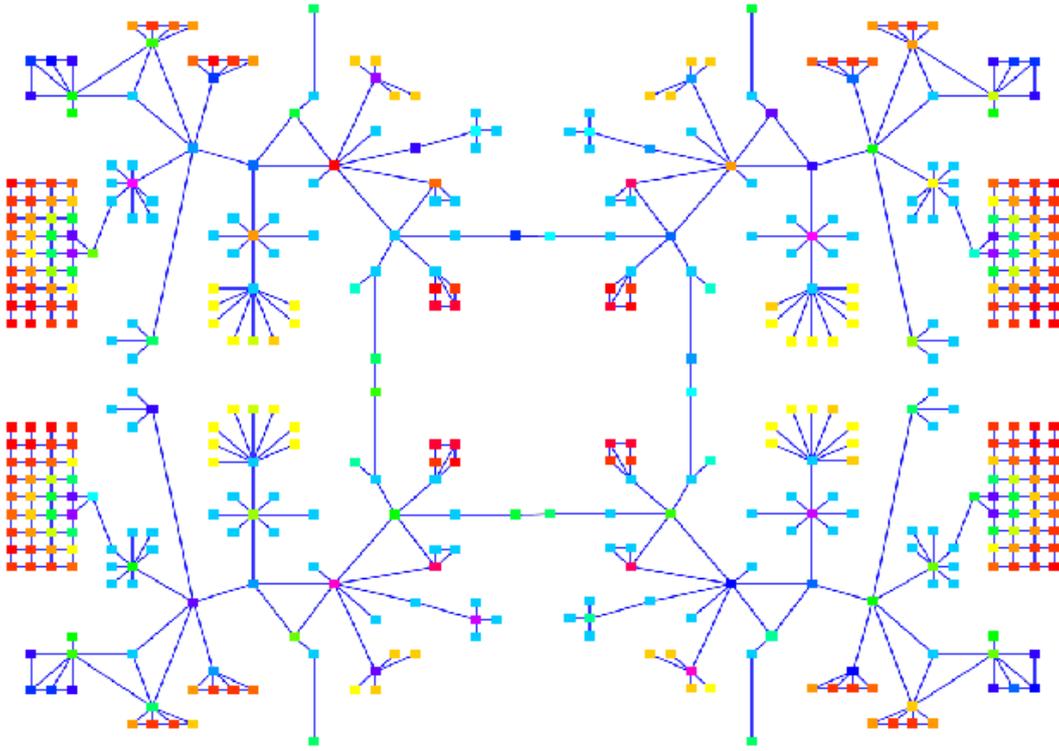This work highlights an important aspect of grid

Figure 6: Model 4 with nodes coloured by their use. This figure shows the resulting node utilisation when hosts issue one request Thaum per unit time. Dark blue and light blue nodes are the heaviest utilised with red nodes being the lightest.

Injection Rate	Model	1 Week			1 Month		
		Storage Util	Super Util	Cluster Util	Storage Util	Super Util	Cluster Util
1/Host/hour	2	54	22	70	57	24	74
	3	53	22	69	57	24	72
	4	53	22	69	56	24	72
2/Host/hour	2	70	39	95	72	42	99
	2	70	39	95	72	42	99
	2	70	39	95	72	42	99

Table 1: Resource utilisation percentages for different types of 'consumer' resources in our model when run over 1 week and 1 month. Model 3 is twice the size of model 2, and model 4 is twice the size of model 3.

planning and decision making. It is desirable to characterise the properties of the whole network by some metric which captures the holistic grid network behaviour. We believe that the moments of the distribution of the agent lifetimes - path lengths discovered - are suited to this.

6 Summary and Conclusions

We have described preliminary work in formulating an agent-based grid simulation, that is similar in structure to our DISCWorld grid architecture. We believe the agent-based approach while less detailed than the network and packet based simulations of other researchers, has its uses and allows some interesting user and resource patterns to be formulated.

In this paper we have restricted our simulations to single coloured jobs which must diffuse through the network to "find" a matching resource. We are presently building more sophisticated mobile job-agents that retain path information and can both learn the best path to a matching resource and which can also pass on this information. We envisage static resource-discovery (or routing) agents that can monitor passing jobs and correlate the best available infor-

mation dynamically to redirect jobs accordingly. We expect that the emergent schedules will display some interesting adaptive properties.

We made the assumption that the use of unconstrained queue sizes would not adversely affect our model. This appears to be true providing the overall amount of resource available in the gridded model system is sufficient to meet the steady state average job demand. Otherwise queues ought to back-up to a maximum limit, which could be accurately simulated by a blocking-job injection model for user agents. This would also remove the necessity for our assumption that agents can write at any time to queues but might also require agents to maintain some partial job store internally, separate from the edge queues.

The simple model presented in this paper tells us that at a naive level our grid model scales linearly. We place N times as many resources **and** users on the network and the overall utilisation metric and user response metric remain much the same. We expect to find some different scaling patterns with smarter agents that do not just diffuse through the network, but take a more intelligent resource discovery action.

Unfortunately, it is not entirely clear from this level of model sophistication whether we can truly test
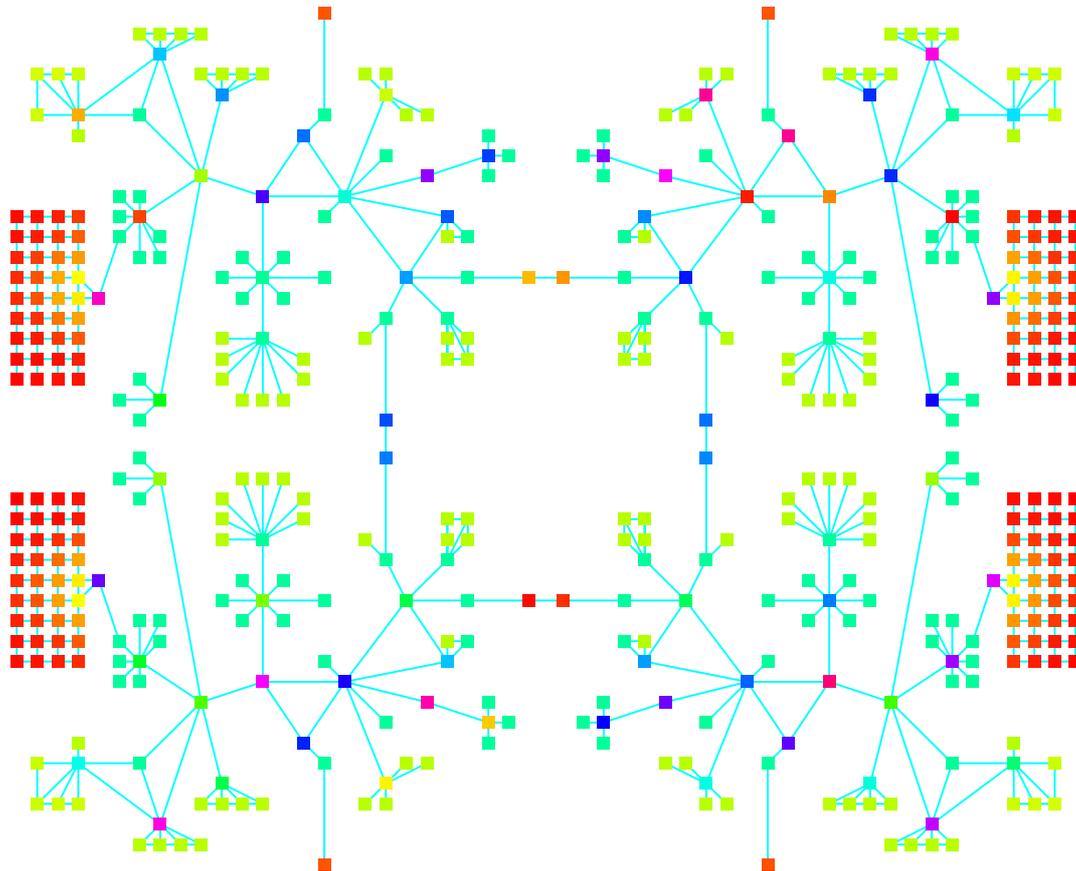
Figure 7: Model 4 with nodes coloured by their use. This figure shows the resulting node utilisation when hosts issue two request Thaums per unit time. Dark blue and light blue nodes are the heaviest utilised with red nodes being the lightest.

the grid hypothesis yet. Nevertheless we believe we have identified the two most useful metrics to consider - namely a measure of resource utilisation per unit time and a measure of mean user-job response time - both averaged over the entire model. We believe our microscopic agent-based approach has promise for an investigation of more sophisticated grid scenarios involving multiple job resources and different job lengths.

We will be interested to compare our model simulation with specific grid patterns and phenomena. We anticipate that the scheduling and resource discovery components of the model will be the most important ones to generalise.

Acknowledgements

References

R. Buyya, & M. Murshed (2002), GridSim: a Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, *Concurrency Computat. Pract. Exper.* 2002; **14**:1175–1220.

I. Foster & C. Kesselman, (2003), *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, ISBN 1558609334.

C. Girault & R. Valk, (2003), *Petri Nets for Systems Engineering*, Spring Verlag, ISBN 3-540-41217-4.

K.A. Hawick & H.A. James & A.J. Silis & D.A. Grove & K.E. Kerry & J.A. Mathew & P.D. Coddington & C.J. Patten & J.F. Hercus & F.A. Vaughan,(1999), DISCWorld: An Environment for Service-Based Metacomputing, *Future Generation Computer Systems* **15**, 623 (1999).

K.A. Hawick (2004), GraViz A Graph Visualisation Tool, unpublished.

Hewlett-Packard Company, (1994), Standard Template Library Programmer's Guide, Available from http://www.sgi.com/tech/stl Last visited September 2005.

C.A.R. Hoare, (1985), *Communicating Sequential Processes*, Prentice Hall International

R.N. Ibbett & P.E. Heywood & F.W. Howell , (1996), HASE: A Flexible Toolset for Computer Architects, *The Computer Journal*, Vol 38, (10).

T. Pratchett, (2000), *The Color of Magic*, HarperTorch, ISBN 0061020710.

A. W. Roscoe & C.A.R. Hoare , (1988), The laws of Occam programming, *Theoretical Computer Science*, 60, Issue 2 (September 1988), pp 177 – 229, ISSN:0304-3975.

A.R. Robertson & R.N.Ibbett , (1993), HASE: A Hierarchical Architecture Design and Simulation Environment for Computer Architects, *UKSS '93, Keswick, UK, UK Simulation Society, 1993*.

A. Sulistio & G. Poduval & R. Buyya & C.-K. Tham, (2005), Constructing A Grid Simulation with Differentiated Network Service using GridSim, *Proc. of the 6th International Conference on Internet Computing (ICOMP'05)*, June 27-30, 2005, Las Vegas, USA.

A. Sulistio & C.S. Yeo & R. Buyya, (2004), A Taxonomy of Computer-Based Simulations and its Mapping to Parallel and Distributed Systems Simulation Tools, *Softw. Prac. Exper.* 2004; **34**:653–673.

Sun Microsystems, (2005), Java Development Kit (JDK) 1.5, available from http://java.sun.com. Last visited September 2005.

University of Chicago, (2005), Globus Toolkit, available from http://www.globus.org. Last visited September 2005.