Res. Lett. Inf. Math. Sci., 2005, Vol.**7**, pp101-115
Available online at http://iims.massey.ac.nz/research/letters/

Managing Community Membership Information in a Small-World Grid

K.A. Hawick and H.A. James
Computer Science
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

As the Grid matures the problem of resource discovery across communities, where resources now include computational services, is becoming more critical. The number of resources available on a world-wide grid is set to grow exponentially in much the same way as the number of static web pages on the WWW. We observe that the world-wide resource discovery problem can be modelled as a slowly evolving very-large sparse-matrix where individual matrix elements represent nodes' knowledge of one another. Blocks in the matrix arise where nodes offer more than one service. Blocking effects also arise in the identification of sub-communities in the Grid. The linear algebra community has long been aware of suitable representations of large, sparse matrices. However, matrices the size of the world-wide grid potentially number in the billions, making dense solutions completely intractable. Distributed nodes will not necessarily have the storage capacity to store the addresses of any significant percentage of the available resources. We discuss ways of modelling this problem in the regime of a slowly changing service base including phenomena such as percolating networks and small-world network effects.

Keywords: small-worlds; computational Grid services; online communities; sparse matrices.

1 Introduction

A computational grid [2] is a collection of computational resources connected via a high-speed computer network that are able to share in the solution of complex scientific problems. There have been a number of popular grid projects across the US (e.g. [6]), Europe (e.g. [5, 14, 15]) and Australasia (e.g. [8, 7]).

Grid initiatives seek to make the inter-operation of distributed computational resources more efficient. This can take the form of hiding the location of servers from users, or providing efficient abstractions to scientific instruments available through the Internet. In addition grid projects make the transportation of data between the distributed computing sites very much easier. Some grid projects such as [6] have defined a large number of standard instrument and data interfaces that are able to be applied to resources on the Internet. While the approaches taken by different grid projects differ, most have at least some kind of *service*. Discussed at length in section 2, services are typically the fundamental units of execution in grid environments. We make a distinction between The Grid as a global entity and a general computational grid infrastructure. Whereas the Grid is a long-term specific project, the grid refers in general to any software that aids the use of computational resources in a wide-area environment. Different projects have attempted to solve varying aspects of the wide-area distributed computing problem, such as fault tolerance and parameterised simulations environments.

Based on the ubiquity of the Internet, grid computing environments promise to permanently change the way in which large-scale computing will be conducted in the near future. Scientific and commercial users are becoming excited about the facilities that the global grid offers. Our vision is for a Global Grid infrastructure in which user communities are able to define and use their own representations of data and services as if they are the only users of the grid. Communities could either have the same or different themes, as discussed in section 2. Communities may overlap in the services they utilise. However they should be able to use those services as advertised by any other community, whether the theme is the same or not, providing they either know its name or its functionality.

In our prototype grid infrastructure, named DISCWorld [8], data and services have associated *metadata* and a concise representation format [9] which can be used reason about services. Tasks are comprised of a *graph* of services; the graph can contain a limited number of cycles and can also contain nested sub-graphs. Services reside on hosts; while some services may retain a small amount of location-awareness information the majority of such information is held at a host level. In order to schedule graphs of DISCWorld services [10] the relevant services must be located on hosts potentially across the Internet; the problem of resource discovery is the main topic of this paper.

In section 2 we describe the concept of a service and the problem of resource discovery more fully. We discuss issues surrounding the representation of services and introduce the particulars of our model in section 3; a reference representation of our system is described in section 4. Some of the issues involved in representing a grid the scale of the Internet are described in section 5. To make the problem more tractable we introduce the concept of a small-world network in section 6.

2 Services & Communities

In this section we elaborate upon what grid **services** are or could become on a global Grid system and outline a notation for describing how they can be uniquely identified and grouped. A service on the global grid can be considered in many ways. In general a service is best thought of as some action a server performs on behalf of a user request or query. The request could in fact come from another server if servers act as peers on the grid. Some typical services might involve running a parameterised program to generate some data or storing a data item or indeed just retrieving a pre-existing data item. With such a definition web servers are essentially supplying a grid service, and indeed the web provides the all important infrastructure for any more sophisticated grid services to function in.

A number of grid projects are exploring the sorts of prototype services that are of use to distributed users. Generally it is easier to discuss the properties of these in an applications-specific context. For the purposes of this paper and the model we describe, we assume that services have a distinct name, some well-defined parameters, and at least one server that offers the service to clients. Services do not exist in isolation - they can be typically chained together according to the data types of their respective outputs and inputs. They can also be typically grouped together according to the communities that develop and use them. For example, there is an established community of image processing users or geographic information systems users and developers already on the WWW. These communities share some common background, interests and most importantly jargon and naming conventions.

Figure 1 shows the conceptual notion of a global grid of many services, grouped together in various ways. One of the key ideas of this paper is that the "clumping" or grouping together of services within sub-communities drastically affects the relationships and hence connectivity properties of a graph of all services. Using this observation we discuss various data structures with which to model service relationships and indeed to compute clustering and graph-distance properties. The various quantitative regimes that arise allows us to propose various ways for grid servers to self-model their behaviour in dealing with service management.

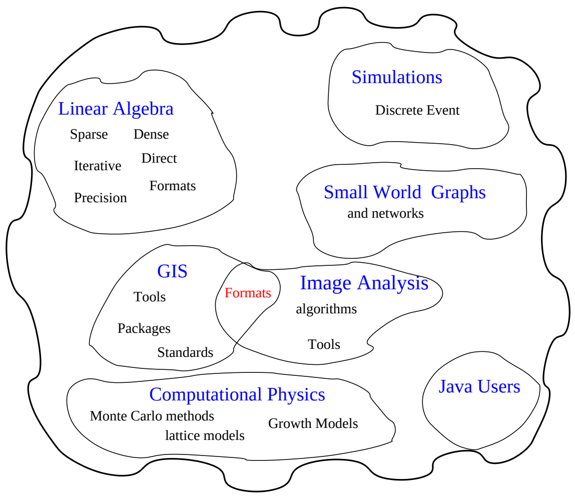

Figure 1: Global space of interacting and overlapping communities - each will have its own set of known, preferred services. They may overlap with those of other communities - and may have different naming conventions.

In this paper we consider the problem of representing the Grid as a system comprising of a number of discrete services N_S, a number of servers N_H each of which can host one or more services, and a group of user communities N_C. Generally $N_S >> N_H > N_C$. At any given time there will be some number of user processes or requests (N_U) outstanding for services to be run. There is no strong relationship between N_U and the other numbers, although we might assume that $N_U \sim N_H$ at any particular time and that $N_U >> N_C$

The problem is essentially the maintaining of information on which servers host the relevant services that are required by different communities. This is analogous to the problem of finding hosts that has been solved through the use of Domain Naming System (DNS) in the Internet. However, unlike the DNS solution, where hosts may have different names through the use of *aliases*, we are not able to mandate a single naming authority for the different user communities on the Internet. We must be satisfied with the restriction that each application community may have their own name for a given service, and perhaps even different names for the same service may be used *within* the same community.

One of the solutions proposed to this problem is allowing each user community or the groups within the community to make and use their own naming conventions, and have those service names ratified by some sort of annual general meeting through the use of arbitration. This model has been successfully used in the botany field in, for example, the naming of a new insect. Clashes however, occur especially since insects like other impure names are laden with conventions and sometimes political connotations. It it entirely possible for it to emerge that two sub communities have been using two different names for the same service for a long time. This must later be resolved. Services on the grid will have the same naming problems as humans do but the clashes must be resolved at a timescale that may be dictated by the Grid and the computers involved (eg milliseconds for the global Grid) rather that at human timescales (eg months or even years).

Resources on the global Grid include services and data items. We assume each resource can be identified by a specific name - pure or otherwise. Users are assumed to have unique names. Services instances must have a unique name although it may later turn out that two services are in fact the same. Servers or hosts must have unique names and in general these hostnames are superposed on the Internet number/location information of a device.

Clumps of services can be (loosely) named as communities which need not have definite unique name although this may help if it is part of the metadata accompanying each service. In this paper we assume that we can and do assign unique names to communities, although two communities of initially different names may merge.

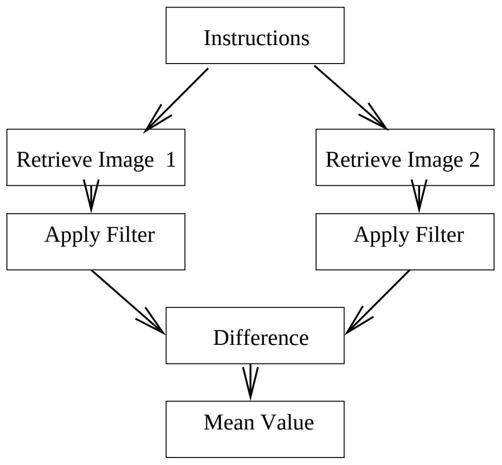

Figure 2: A simple image processing task DAG.

We consider tasks to consist of a finite chain of services linked together with some data being passed between the services [10, 11, 9]. Figure 2 shows a simplified example of service dependencies. The task graph is for an image processing example. Each box represents a service in a grid environment, and the arrows indicate task dependencies or dataflow. We would expect to find a typical task consisting of services from one user community clump, however it is not impossible that a task could span communities. Consider, for example, a user in the GIS community who most often relies on services found within that community. Occasionally the user might wish to use services from the Simulations or Linear Algebra communities. On these occasions the services used by the task will span clumps. Our system is able to organise the efficient partitioning and scheduling of complex task graphs across distributed computing nodes [10].

A key idea is that services are interdependent upon one another. Processing services may need to chain together to satisfy a compound processing request and they may need short or long term storage services to retrieve or deposit inputs to and outputs from calculations. A global Grid needs efficient and practical means for services or resources to **discover** one another either by explicit name or by matching metadata, including data types and processing command arguments.

We might suppose that a metadata matching service of low or high simplicity is some capability of all services on our Grid and forms their means of interacting or "knowing about" one another.

3 Services Cross-Awareness

In this section we discuss how the graph of cross-awareness or service discovery knowledge can be represented as an adjacency matrix.

Example Linear Algebra Services	**Data Types**	**Resource Requirement**
Store a Matrix as part of a calculation	M	Temp storage
Store a Vector as part of a calculation	V	Temp storage
Store or retrieve a particular matrix	M	Long term storage
Extract particular value from a matrix	M :	Compute
Dense Matrix-Vector Multiply	M, V: V	Compute + Temp Storage
Dense Matrix-Matrix Multiply	M, M: M	Compute + Temp Storage
Dense Matrix Invert	M : M	Compute + Temp Storage
Dense Matrix Solve	M, V: V	Compute + Temp Storage
Sparse Matrix to Dense Matrix Expand	S: M	Compute + Temp Storage
Sparse Matrix Conjugate Gradient Solve	S, V: V	Compute + Temp Storage

Table 1: Linear Algebra Grid Services

Example Imaging Services	**Data Types**	**Resource Requirement**
Store an Image as part of a calculation	I	Temp storage
Store or retrieve a particular Image	I	Long term storage
Store or retrieve a particular Kernel	K	Long term storage
Extract particular pixel from an image	I :	Compute
Apply a kernel to an Image	I, K : K	Compute + Temp Storage
Apply well-known operation (eg FFT) to image	I : I	Compute + Temp Storage

Table 2: Image Processing Grid Services

Tables 1 and 2 shows some example services for user communities of linear algebra and image processing. Just creating these fictitious simple services highlights many of the important issues and obstacles to a grid service oriented world. We suppose there is some metadata associated with each item of data. Data type is either assumed to be a bulk item worthy of special consideration such as a Matrix, a Vector, an Image, an image-transform specification Kernel, or a scalar which can be packaged up in the metadata. Some operations require temporary or essentially anonymous temporary storage that is only needed for the duration of a processing request, whereas other operations require some storage service that itself requires negotiation to ensure transmission of retrieval metadata as part of the service. In these examples we have glossed over details connected with the size of the particular data items or the scope of the tasks. For small images or matrices perhaps many computer hosts can offer the requisite services. For very large matrices only supercomputers can offer a matrix solver service. There will of course be many specialist variations of a generic service with different algorithms or options available. It is no trivial matter to set up a schema for all the metadata and indeed the type space problem[3] itself is a difficult one to approach. We do not address these issues in this paper but continue with discussions of the more general naming and discovery issues.

For simplicity we imagine services like those in tables 1 and 2 as having some pure name (that is unique to our whole system). Each service is labelled by a unique integer k. We further suppose that we can uniquely label our servers or hosts by a

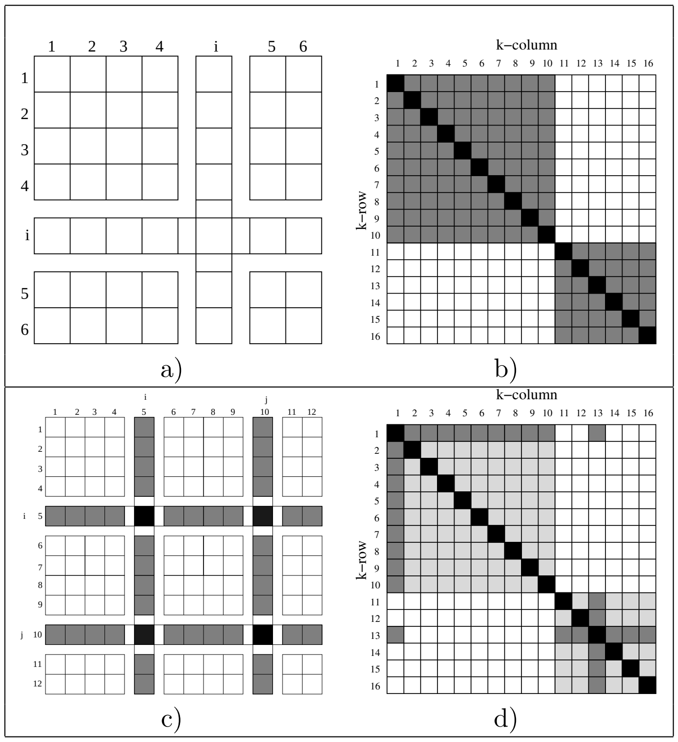

Figure 3: a) Insertion of a row i into a 6×6 adjacency matrix, requires insertion of both a new row and a new column. b) 10 Linear algebra services running on one host for community One and 6 Image processing services running on a second host for community Two. Each service knows about itself (the diagonal elements) and the lighter shaded blocking shows that each community is fully and densely connected. Every service element in each community knows about every other service in that community. the two communities are disparate however. c) Permutation (swap) of two rows i and j requires swap of their corresponding columns and also the individual swap of their cross over elements. $i, j \mapsto j, i$ and $i, i \mapsto j, j$. d) Service 1 is designated as a gateway node in community One and node 13 has been designated gateway node for community two. Dark gray shading indicates gateway node knowledge of other services (rows) and services knowledge of the well known gateway (columns). In addition services 1 and 13 know about one another.

unique integer j and that each jth host can refer to the particular services it offers by a unique integer i.

We have: $k = 1, 2, ..., N_S$; $j = 1, 2, ..., N_H$; and $i = 1, 2, ..., N_{H_j}$. This then allows us to generate a unique integer k' for each service instance. We can construct various number to metadata or name mapping functions or operators providing we have no ambiguities.

Figure 3a) emphasises the fact that the adjacency or “knows-about” matrix is square and any particular service element has a corresponding row and column. The diagonals can be treated either as always empty or always as a self-knowledge element. In using the adjacency matrices in calculations it is often easier to fill the diagonals, but self knowledge of a service is somewhat obvious as shown in figure 4 and obscures the in- and out- degrees of the nodes which correspond to the “knowledge of” elements in our matrix. We adopt the convention that matrix element i, j corresponds to knowledge of service j by service i.

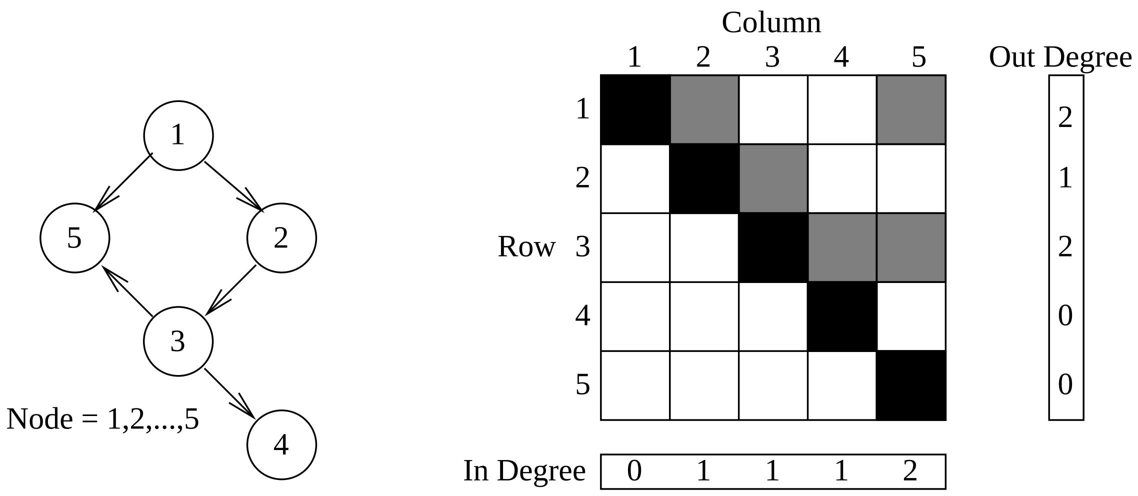

Figure 4: Adjacency matrix for a simple task graph of five nodes. The row sums (excluding diagonals) give the out degree and the column sums (excluding diagonals) give the in-degree of the nodes. Degree corresponds to knowledge in our service matrix.

Consider an example grid of two communities each with one host. Host 1 offers Linear algebra services (numbered for the example as $1, 2, ..., 10$); Host 2 offers the 6 Image Processing Services. We can describe this information in a matrix representation as shown in figure 3b). The adjacency matrix shows the graph relationship "knows about", where service in row k_r "knows about" the service in column k_c. For simplicity in this example make each service have only one instance, but we can construct the matrix for instance-indices (k') rather than service-indices (k).

The sparse blocked matrix representation that arises is shown in figure 3b) and can be analysed for various properties including scaling and connectivity. It is sometimes useful to consider the decomposition of the adjacency matrix $A = L + D + U$ into diagonal and upper and lower triangular components. In our notation the upper triangular part denotes knowledge of node j by i. The diagonal can be factored out as it only denotes self knowledge.

It is likely that a particular host-server will not just offer unrelated services but will offer services associated with a particular community. In many cases hosts **will** serve **one** community, but this is not necessarily so and so it may be more useful to group service instances by community rather than by host. We might optimistically assume that communities are sufficient for our analysis as a particular service will have a single well known name within a particular community. This is not always the case however and we have sometimes to drop back to the case of unique service names within a particular host - which generally will be the case for operational reasons.

We can transform our matrix representation by permuting rows to group like-services together by community or host. Providing our number/naming identification system, is pure this does not affect the "who knows about who" relationships, but it can make the matrix easier to interpret visually. Figure 3c) shows how two services can be swapped or permuted. Both individual rows and columns must be swapped and the individual cross elements must be individually swapped also. Repeated use of this permutation operation allows us to group services any way we wish - by community or by host. Both groupings may be useful for calculation of properties of our service grid.

4 Adjacency Matrix Structure

The awareness or resource discovery matrix representation illustrates a number of ways for services to be aware of one another. A fully dense structure would indicate that every service knew about every other service. This is the way the Internet started. One of us remembers working on a mainframe in Edinburgh during the early days of the global Internet, and having to receive a "hosts file" periodically from the USA giving the names and addresses of every other host that was registered on the Internet. Obviously this was untenable for the Internet hosts' cross-awareness knowledge as the number of hosts grew. The Domain Name Service based on a tree like structure of naming authority and responsibilities arose. A tree like approach is another obvious way to group our services. If each community (or host grouping) has a single "gateway service that all services in that community (or host) know about and which acts as the point of registry for all its local services then the structure as shown in figure tree could be used.

Figure 3d) shows how service cross-awareness can be enabled using "gateway" services which are "well-known" within their communities and which in turn "know" the other services within their communities.

Other matrix structures that could arise are mentioned in [13, 1]. They indicate different global properties and structures of the grid. It is certainly possible that different groupings, aliases, permutations, and hosts may exist that serve more than one community. Communities may also require more than one host.

Aliases could arise when two independent groupings realise they are referring to the same service by different names. Strictly this should result in a smaller matrix, but as a temporary measure we might consider a proxy or dummy entry that appears in place of another service. All proxies for a service might would know about the actual service, but the relationship need not be symmetric. A real service need not know whether it is being accessed by another real service or by a proxy. It becomes problematic if a real service is ever removed or disabled when it is being used to support a proxy.

We can categorise the different regimes of service-service awareness as follows: a) Simplest regime every server offers every service - knowledge matrix is dense. b) Simple small community regime - everyone knows about everyone else in their own community - spares block structured matrix. c) Communities of communities regime - where a sparse structure links together dense communities. d) Almost Scalable communities - a sparse structure is applied globally and locally and gateway services have dense knowledge. e) Global scalable system where all parts of the knowledge matrix are sparse and a small-world network links gateways which impose their own internal community dense/sparse structure depending upon size of community.

5 Global Grid Scaling

Our design goal for a resource discovery system is a fully reachable resource knowledge graph, implemented as efficiently as possible. Space efficiency requires avoiding dense table storage, whereas access efficiency requires minimising the number of "hops" across the graph are needed to find a particular service instance. In this section we consider scaling issues - how many servers, services and communities and what regimes are there on the present Internet and use this to speculate how the global Grid of services might develop.

Table 3 shows the number of web pages as reported by the Google search engine for search terms corresponding to the application domains in figure 1. The first column shows how many web pages were reported as containing just the application domain name, while the second has the number that purported to contain some

Application Domain	**Web Pages**	**RAS Pages**
Linear Algebra	1,690,000	19,500
Simulations	4,320,000	140,000
GIS	1,880,000	147,000
Image Analysis	6,880,000	698,000
Computational Physics	1,240,000	97,000

Table 3: Number of web pages reported by Google (http://www.google.com) to contain the name of the application domain and and also the string "remote access services". Searches completed on 10th June 2004 out of 4,285,199,774 web pages indexed by Google.

extra search strings. While the numbers of web pages returned in each search is only representative, it does serve to illustrate the huge number of not only communities on the Internet, but also the number of members of each community.

It is safe to assume that the number of user communities and also the membership within each community does not remain static for too long. For this reason the major problem in maintaining information as to which services are available on a grid is that of ensuring it is consistent. The two logical extremes are to maintain all the information that is known about the services available on the system, and to maintain no information about services. One one hand it is infeasible to require all updates to the services one offers be broadcast to all members of the community but on the other hand it is also infeasible to require services be found each time they are used.

A scaling strategy that reduces the number of links while maximising the reachability is desirable. Small-world approaches offer some exciting possibilities for such a solution.

6 Small-Worlds Solutions

Traditional network models use regular grid-ed or tree-based structures for representation, where nodes in the network graph are linked only to their nearest neighbours. We make the observation that typical hosts on the Internet maintain active knowledge of only a very small number of hosts that are outside their direct local area. In a graph context this counts as a nearest neighbour relationship. Bearing in mind that DNS organises the Internet into a tree-based hierarchy we also find that some distinguished hosts also maintain knowledge of other hosts that are not classified in the local area network (or hence not strictly nearest neighbours). Through the use of DNS hosts have the address of a node that is higher in the tree-based hierarchy of domain names, in order to discover hosts from outside their local network.

If we consider how the representations of regular grids or tree-based hierarchies can be modified then one minor modification is to add a number of direct links between different links in the graph. As this is done, the graph will enter the realm of small world networks [16]. Adding links is not the only way a graph can exhibit small world effects: random links can be removed and also random links can be rewired. These modification strategies are reviewed in [12]. In this paper we restrict ourselves to only considering the addition of links.

Community clustering is one mechanism that promises to reduce the effective all-pairs distance for entities *within* a community. However we observe that even if we allow the members of a community to know about all the other members (using $O(n^2)$ graph links) then there are still latency effects when task requests go outside the community to provide services.

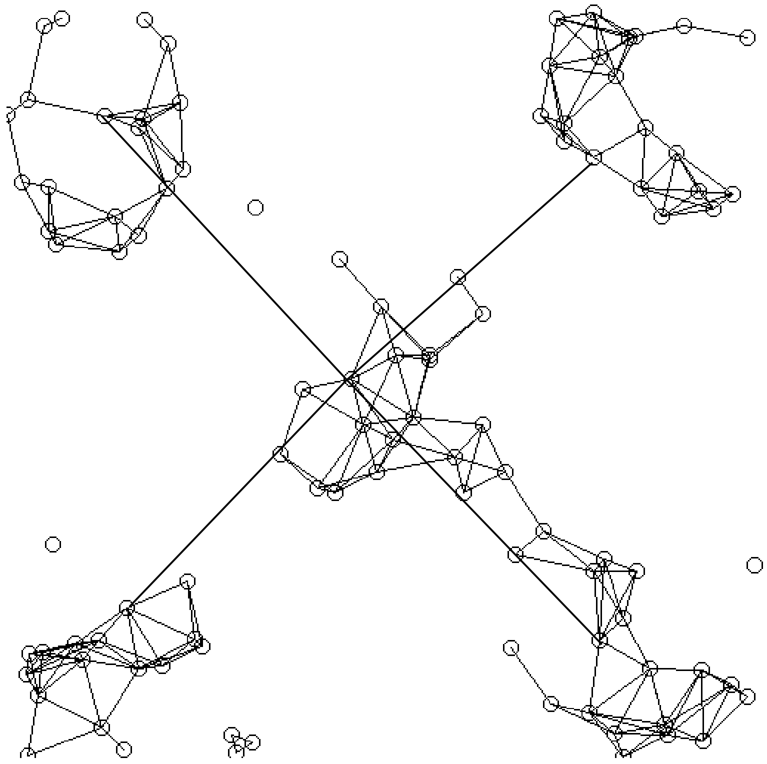

Figure 5: A simulated space of five “communities” which may connect by chance, or by addition of explicit shortcuts. Some services are isolated not having been registered properly within their communities.

Figure 5 shows how five example communities have been cross linked with some explicit small-world shortcut links. Each community may in fact have some straggler service s that have developed separately and are not yet fully reachable. Gateway nodes will help this, particularly if the gateways are linked in their own small-world sub-structure.

Main criteria for performance is pairwise distance. A good metric is the all-pairs distance for a graph. We can investigate this for small service discovery graphs, however since algorithms for computing this scale as N_S^3, we are limited practically to graphs around a few thousand nodes, even with a supercomputer assistance. We are unable to directly simulate realistic numbers of fully connected services. In practice however the global grid **will** consist of a number of loosely connected communities each one of which may be more strongly connected internally.

The shortcuts we can deliberately place on an otherwise loosely connected resource graph can make a dramatic difference both to its stability in terms of full-connectedness, and also to the speed of access or reachability of a particular service.

Figure 7 shows the all-pairs distance measured in a system where each service connects (knows about) each other service with probability p. Figure 8 shows the mean cluster size as p is varied. Figure 6 shows the mean number of clusters.

Figure 5 shows the way in which communities may be clustered and connected to each other. We can see that within communities there is essentially a fully-connected graph of resources. This signifies that that within a community there is almost complete knowledge of the resources that are available. However between communities resources are not as easy to access. This means that when a resource

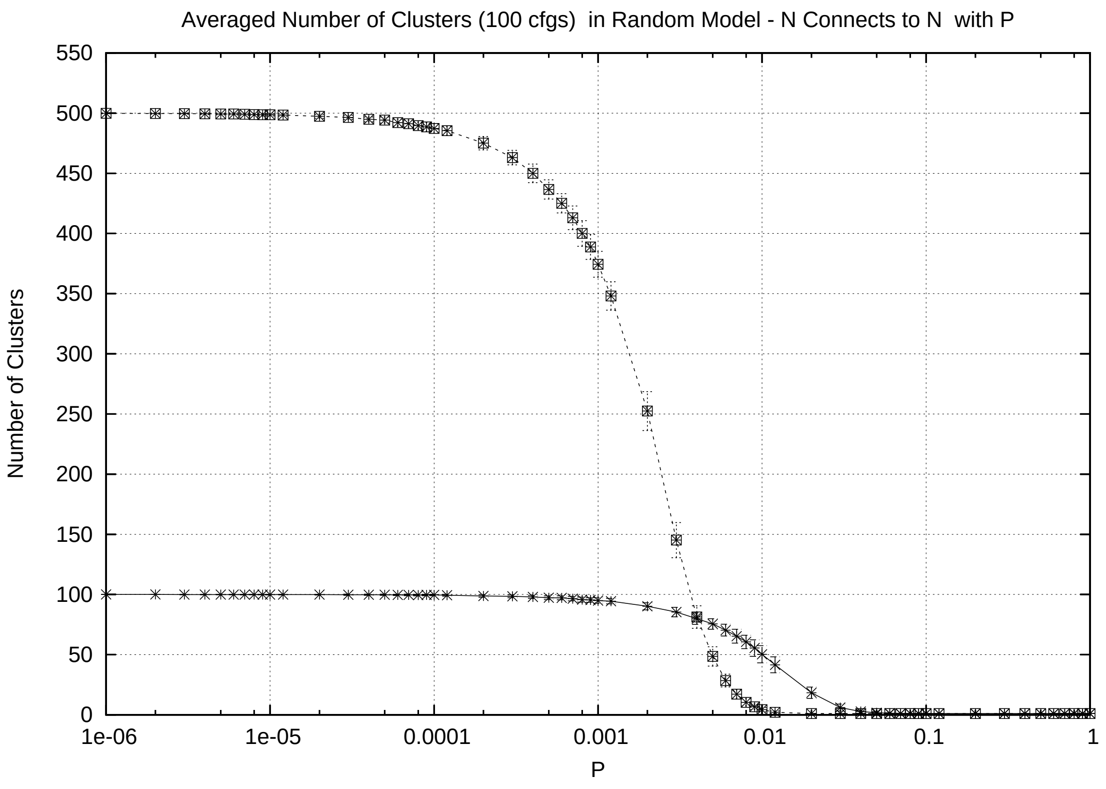

Figure 6: Mean number of clusters in a system where each service connects (knows about) each other service with probability p.

inside one cluster needs to find a matching service that is not inside its own cluster, the searching will take longer on average and also accessing the remote resources will involve more cross-community hops. We believe this is the way in which more and more communities will arrange themselves on a global grid system.

The Small-world approach can considerably augment the performance of our resource discovery process. The rationale for small-world modifications is quite simple: services, and the hosts (entities) they reside on will know only about a small number of other entities by default. The typical mechanism that is used in representing complex task graphs is simply to name the required entity, not where it can be found [10]. Thus to find another entity a series of resource discovery requests will need to be made, increasing with time as the entity is further away in terms of graph distance. If a random link can point either directly to the required service, or more likely, to a host that knows about the required service, then the search time will be shortened. The search time has a physical property in the latency of searching. We assume that the bandwidth that is achievable by the network between any two hosts or services is constant but not the same. Thus the major opportunity for enhancing the performance of any system is through eliminating any wasted searching (resource discovery) or access time.

Figure 7 shows how the mean all-pairs distance, calculated using Dijkstra's algorithm, changes with the probability p that a random direct link is made between any two random entities in the grid. This is directly analogous to the case under consideration where an entity randomly points to another entity in the graph. Ignoring any DNS-type resource discovery hierarchy, and only considering the addition of random links between services or hosts, the higher the probability of an

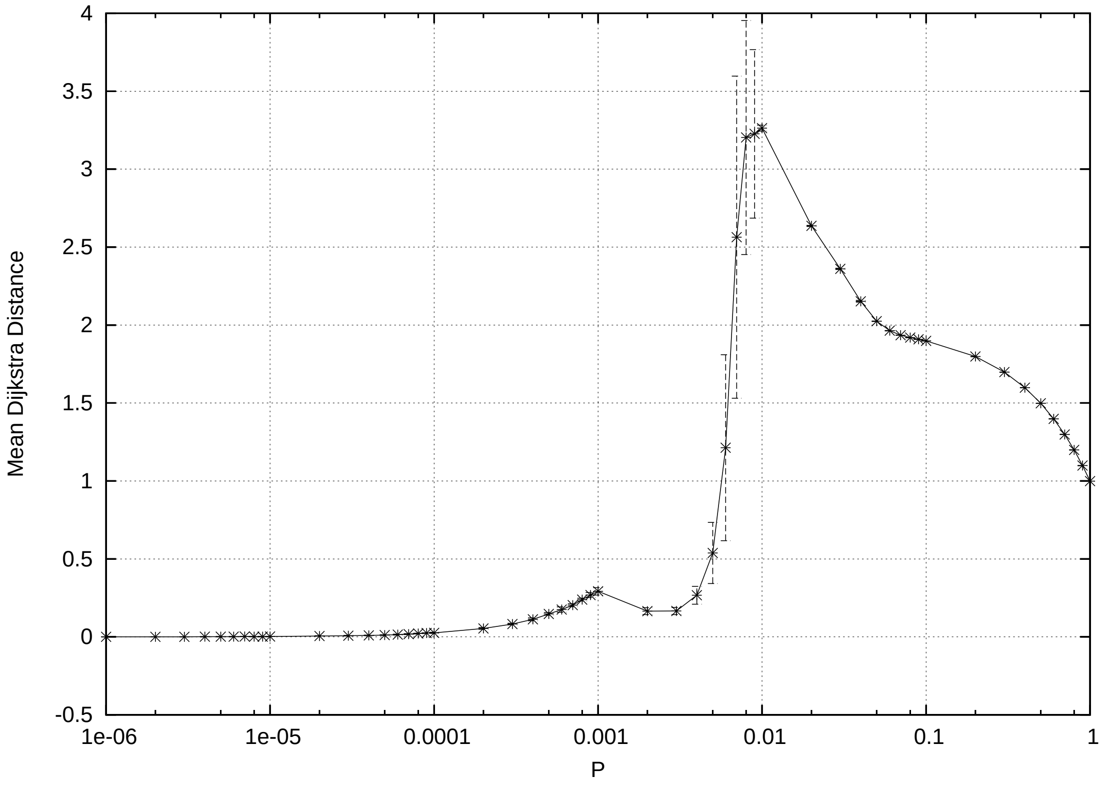

Figure 7: Mean All-Pairs distance in a system where each service connects (knows about) each other service with probability p.

entity pointing to another entity increases both the mean cluster size (figure 8) and also the mean Dijkstra distances (figure 7) until a point is reached such that the graph consists of one cluster. In practice having some cluster, implying that every entity knows about each other entity, is impractical for space and efficiency reasons. However we find that at lower connection probability (between $0.001 \leq p \leq 0.01$) the searching efficiency is much improved.

If we now allow the DNS-type resource discovery hierarchy, the main consideration for performance is still the mean pairwise distance between entities in the graph. The pair-wise distance corresponds to the number of queries that must be made for a graph entity to either find, or communicate with, another entity. The higher the pairwise distance the more intermediate hosts must be used for the communication and the higher the latency will be in communications between the original source and destination.

Being able to successfully find another service or hosts on the grid is the primary aim of the model. In addition the system architecture needs to maintain a model that is correct and fault tolerant. It would be disastrous if the system were to suffer segmentation should a single host be unavailable. In computer networks fault tolerance is provided by the addition of redundant links. Grid systems are built on top of the Internet by necessity; fault tolerance is provided by the addition of direct links between grid-ed hosts - it is left up to the Internet routing protocols to organise the most efficient physical way of routing messages.

As a secondary, but still quite important concern, the issue of evolution must be addressed. In terms of the model, this refers to the ability of the system to adjust efficiently to the changing conditions of the global grid environment. Changes will

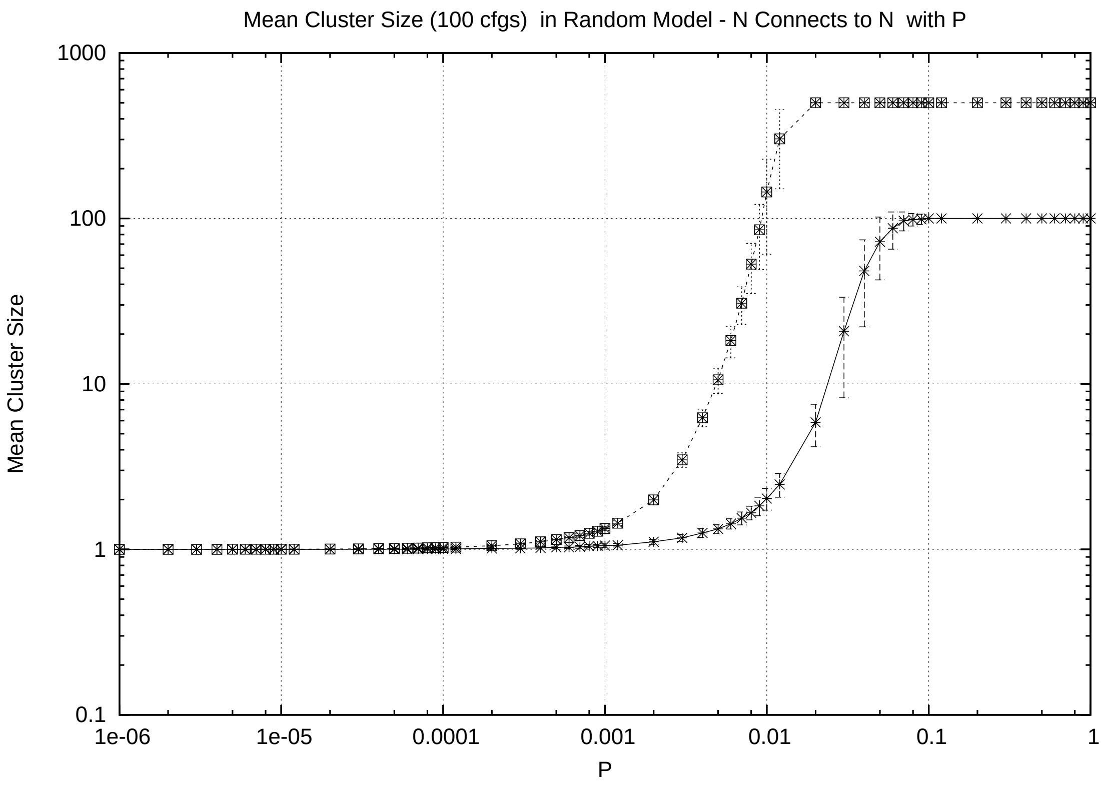

Figure 8: Mean cluster size in a system where each service connects (knows about) each other service with probability p.

typically involve the addition and removal of hosts, and of course the addition and removal of the services that reside on the hosts. Services in the DISCWorld environment are typically written in Java; the language's portability means that byte codes can be run on virtually on any platform. This compounds matters as our model must be able to cope with services that have been migrated or even copied between hosts in the grid.

A necessary feature of our model's implementation is the ability to react efficiently to the addition or deletion of hosts, services or communities. We are aware of the equivalent problem in evolving computer networks, the so-called *counting to infinity* problem. For this reason we find that the addition of new random cross-community small-world links can help aid the system in evolving rapidly. This technique is only possible due to the fact that the global grid consists of a number of loosely connected communities, with strong internal interconnections. The cross-community links are used in the regular systolic resource discovery algorithm to discover new and changed resources, where changes are able to percolate through the system more rapidly due to the randomised small-world links.

7 Conclusions and Summary

We have discussed the significant problem of resource discovery in global grid systems. We note there is an ever-increasing number of Internet-based resources, a good percentage of which will eventually be connected to a grid system quite possibly arranged as well-connected groups of communities. Such communities will have links to other communities. It is infeasible to maintain a completely connected matrix of available resources in one computer's memory, let-alone have this information replicated across every participating host on the grid. Thus a sparse storage mechanism is one practical storage option.

Representing the grid computing infrastructure as a small-world graph allows us to maintain a tractable searchable information resource that effectively minimises the searching time while maintaining a practical limit on the number of other resources that must be known about to accomplish the searching. Our prototype simulation of this model features uni-directional links that point from a source to a destination node in the graph. In future work we will adapt the model to incorporate bi-directional data pointers as afforded by the DRAM abstraction [10].

The global Grid is changing the way we approach all sorts of computing problems. We believe that small-world analysis of the emerging service infrastructure is truly important for achieving a genuine global village of service communities.

References

[1] Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., van der Vorst, H. "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", SIAM, Philadelphia, PA, 1994.

[2] Berman, F., Fox, G.C., and Hey, A.G.J. (Eds), "Grid Computing - Making the Global Infrastructure a Reality", Wiley, 2003, ISBN 0-470-85319-0.

[3] Cardelli, L. "Type Systems", in Allen B. Tucker (Ed.): The Computer Science and Engineering Handbook. CRC Press, 1997, ISBN: 0-8493-2909-4. Chapter 103, pp 2208-2236.

[4] E.W. Dijkstra. "A note on two problems in connexion with graphs" Numerische Mathematik, 1:269–271, 1959.

[5] EUROGRID Project. http://www.eurogrid.org

[6] The Globus Alliance. http://www.globus.org

[7] GridBus Project. http://www.gridbus.org

[8] Hawick, K.A., James, H.A., Silis, A.J, Grove, D.A., Kerry, K.E., Mathew, J.E., Coddington, P.D., Patten, C.J., Hercus, J.F. and Vaughan, F.A., "DISCWorld: An Environment for Service-Based Metacomputing," *Future Generation Computing Systems (FGCS)*, 15:623–635, 1999.

[9] Hawick, K.A., James H.A. and Mathew, J.A., "Remote Data Access in Distributed Object-Oriented Middleware" in J. of Parallel and Distributed Computing Practices 3, 3-19 (2002).

[10] James, H.A. "Scheduling in Metacomputing", PhD Thesis, Department of Computer Science, The University of Adelaide, Australia, 1999.

[11] James H.A and Hawick, K.A., "Data Futures in DISCWorld" in Proc. of High Performance Computing and Networks (HPCN) Europe 2000, Amsterdam, May 2000.

[12] James, H.A., Scogings C.J., and Hawick, K.A., "A Framework and Simulation Engine for Studying Artificial Life" in Res Let in Info Math Sci, Massey University, May 2004.

[13] Press, W.H., Teukolsky, S.A., Vetterling, W.T, and Flannery, B.P., “Numerical Recipes in C. Second Edition”, Cambridge University Press, 1988.

[14] UK e-Science Grid Support Centre. http://www.grid-support.ac.uk

[15] UNICORE: Uniform Interface to Computing Reseources. http://unicore.-sourceforge.net

[16] Watts, D.J., “Six Degrees - The Science of a Connected Age”, Pub William Heinemann, 2003, ISBN 0-434-00908-3.