Synthesizing Realistic Computational Grids

Dong Lu Peter A. Dinda
{donglu,pdinda}@cs.northwestern.edu
Department of Computer Science, Northwestern University

Abstract

Realistic workloads are essential in evaluating middleware for computational grids. One important component is the raw grid itself: a network topology graph annotated with the hardware and software available on each node and link. This paper defines our requirements for grid generation and presents GridG, our extensible generator. We describe GridG in two steps: topology generation and annotation. For topology generation, we have both model and mechanism. We extend Tiers, an existing tool from the networking community, to produce graphs that obey recently discovered power laws of Internet topology. We also contribute to network topology theory by illustrating a contradiction between two laws and proposing a new version of one of them. For annotation, GridG captures intra- and inter-host correlations between attributes using conditional probability rules. We construct a set of rules, including one based on empirical evidence of OS concentration in subnets, that produce sensible host annotations.

1 Introduction

The goal of grid computing [18, 16] is to give users easy access to arbitrary amounts of computational power, connectivity, storage, and services within large wide-area distributed computing environments. Middleware such as Globus [17] and Legion [20] simplify using remote resources within a computational grid. To help users find resources, these systems typically provide some form of a grid information service (GIS) such as Globus MDS [9]. Resource monitoring tools such as NWS [35], Remos [10], Wren [23], or RPS [12] can be used to gauge the dynamic availability of found resources.

Designing and evaluating such grid middleware demands realistic workloads. For example, we are in the process of designing and building a grid information service based on the relational data model [11, 13]. In this system, users are able to pose complex compositional queries that resemble decision support queries. A typical query might look for a group of machines that use the same OS, together have a certain amount of memory, and that the subset of the network connecting them have some bisection bandwidth. To make these queries fast, we implement them using stochastic search, allowing us to trade off between the number of nondeterministically chosen results returned by the query and the amount of work done in support of it. This tradeoff depends strongly on the structure of the grid: the network topology and the characteristics of the hosts, routers, and links within the topology.

While Smith, et al [30] studied the update and query processes on such grid information, there is no extant work and limited available data on the structure of computational grids. We examined the contents of several running GIS systems. The largest dataset we have found, generously provided by Smith, contains fewer than one thousand nodes. Given the limited data sets, a synthetic grid generator is a necessity. Furthermore, even as more data becomes available, it will continue to be useful to have a parametric source of grids. For example, such grids could be used with simulation toolkits such as GridSim [6], Simgrid [8], MicroGrid [31] and Bricks [1]. to study the benefits of different scheduling techniques. Unfortunately, no such generator currently exists. Synthetic grids could also be useful in simulation studies of overlay networks and peer-to-peer systems [29, 4]. Although the correctness of such protocols do not depend on the underlying topology, their efficiency does depend on topology and on the performance of the end-systems.

In response to this need, we have built GridG, a grid

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, and EIA-0224449. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

SC'03, November 15-21, 2003, Phoenix, Arizona, USA
Copyright 2003 ACM 1-58113-695-1/03/0011...$5.00

generator. Our definition of a grid is an annotated directed graph in which the nodes represent hosts, routers, switches, and hubs, and the edges represent network links. The graph is thus a network topology that extends to the level of hosts. In addition, each node or edge is annotated with information relevant to its use as a part of a computational grid. A grid generator, such as GridG, produces a grid of a given number of hosts. It must meet the following requirements:

- It must produce a realistic network topology. Much is known about the properties of real network topologies: they are connected, and they have hierarchical structures. Furthermore, wide-area network topologies, including the Internet, have recently been found to follow certain topological power laws [15]. A good generator will provide both structure and follow the power laws [32].
- It must generate realistic annotations for hosts and network components. For a host, it should at least provide the architecture type, processor type and speed, number of processors, memory size, disk size, hardware vendor, operating system, and available software. For a link, it should provide the hardware bandwidth and latency. For routers and switches, it should specify the aggregate backplane or switching fabric throughput and latency. It should capture correlations between different attributes (for example, we might expect that memory size increases with processor speed with high probability), and between nearby components (for example, a high speed router is unlikely to be connected only to a few slow links).

The networking community has produced a wide range of topology generators, including random Waxman [33], Tiers [14, 7], Inet [21, 34], etc. These generators either meet the structure requirement or they meet the power-law requirement. GridG starts with the output of a structure-oriented topology generator (we currently use Tiers) and adds redundancy to it in such a way as to make it conform to the power laws. As far as we are aware, this makes it the first topology generator that provides structured topologies that obey the power laws.

GridG in fact directly enforces only one power law, the so-called outdegree exponent power law. Its outputs, however, show obedience to all the other laws as well. In studying the unreasonable effectiveness of the outdegree law, we discovered a new fact: it is either the case that the so-called rank exponent power law is not actually a power law, or that it is more significant than the others. We believe that the former is actually the case, but that over the typical range that is considered, a power law is a useful approximation. At this point, we believe the outdegree law is more significant and that the other power laws can be derived from it.

GridG provides mechanisms for annotating each node and edge. These mechanisms are currently based on user-supplied empirical distributions and conditional probability rules. The rules enforce correlations between different attributes on the same graph element and correlations between different graph elements that are close to each other. Distributions and rules can be determined by measurement. For example, we discovered OS concentrations on all the class C IP subnets we probed with nmap—most subnets appear to have a dominant operating system. These kinds of clustered attributes are very important for Grid modeling, resource allocation and scheduling research, because much of the resource allocation and scheduling work that has proved successful depends on clustered homogeneity of such attributes. We added this optional rule to GridG as an example of capturing correlation between attributes on nearby graph elements. We also added optional rules correlating the different attributes of a host that are not measurement based, but reflect the sensible beliefs most people have of how machines are configured.

Unfortunately, very little is known about the characteristics of a grid or network that are represented by the annotations. We point to the information that we think is necessary to develop models of these little studied network and host characteristics. The successful collection of this kind of data, and the models that could be developed from it are a very exciting research opportunity.

In the following, we begin by presenting the overall architecture of GridG in Section 2. While GridG can apply any number of transformations to produce a grid, there are two core steps: topology generation and annotation. In Section 3, we describe how topology generation is done and demonstrate that GridG conforms to the power laws of Internet topology. Section 4 discusses our insights into those laws, including the apparent contradiction between two of them. Section 5 describes the GridG mechanisms for annotation, outlines the requirements of a model for annotation, discusses the OS concentration phenomenon, and describes the open research questions posed by the need for such a model and how we are attempting to address them. Finally, Section 6 concludes the paper.

GridG version 1.0 is available online at the following URL: http://www.cs.northwestern.edu/∼urgis/GridG.

2 Architecture

GridG is implemented as a sequence of transformations on a text-based representation of an annotated graph. The transformations are generally written in Perl, although this is not a requirement. Figure 1 illustrates how these transformations are composed to generate a grid. Currently, we begin with a structured graph without redundancy that is generated by the Tiers topology generator. The number of networks at each level of the topology is the primary input. This also indirectly specifies the number of hosts. The first transformation enforces the power laws by adding extra links to the graph. The outdegree exponent is main input. The next transformation annotates the graph according to user-defined empirical distributions on and correlations over attributes such as memory and CPU. Additional transformations can be added. For example, we can add clusters to sites on the grid. The final output can then be visualized with DOT, used for GIS evaluation, or for other purposes.

3 Topology

GridG generates topologies comprised of hosts, routers, and IP layer links. In GridG's graphs, nodes in WANs and MANs are routers while nodes in LANs are hosts. Routers have switching capability and several IP interfaces while hosts have computing and storage resources.

Recent research [5, 3, 22] shows that many natural and artificial networks follow the so-called outdegree power law, including such examples as molecules in a cell, the power grid, the World Wide Web, species in an ecosystem, and people in a social group. In particular, recent work [15, 28] show that not only does the Internet topology [15] follow this power law, but also the peer-to-peer sharing overlay network Gnutella [28]. Like the Gnutella network, future grids will be embedded in the Internet topology and thus will likely follow its rules.

3.1 Power laws of Internet topology

Faloutsos, et al [15] identified three power laws and one approximation in their influential 1999 paper. Figure 2 summarizes these laws. (Figure 3 summarizes the symbols used in this paper.) The rank exponent law says that the outdegree, d_v, of a node v, is proportional to the rank of the node, r_v, raised to the power of a constant R. In our examples and evaluation, we choose to parameterize our power laws according to the router-level data in the Faloutsos paper. The parameterized rank exponent law is

Power Laws	Rank exponent	$d_v \propto r_v^R$
	Outdegree exponent	$f_d \propto d^O$
	Eigen exponent	$\lambda_i \propto i^\varepsilon$
Approximation	Hop-plot exponent	$P(h) \propto h^H$

Figure 2: Power laws of Internet topology.

$$d_v = \beta r_v^R = exp(4.395) * r_v^{-0.49} \qquad (1)$$

The omitted constant term does not affect our results and is commonly dropped [2]. Another useful form of the rank exponent power law is

$$r_v = (\frac{d_v}{\beta})^{\frac{1}{R}} \qquad (2)$$

The outdegree exponent law says that the frequency, f_d, of an outdegree d, is proportional to the outdegree raised to the power of a constant O. Parameterizing the law using the Faloutsos router-level data, we have

$$f_d = \alpha d^O = exp(8.52) * d^{-2.49} \qquad (3)$$

A node's ranking is defined in the following way, conforming with the Faloutsos paper. We do a topological sort of the nodes in decreasing order of outdegree. We then assign ranks according to this ordering and the number of nodes in each equivalence class. All n_1 nodes in the class with largest outdegree are assigned rank $r_v = 1$. All n_2 nodes in the class with the second largest outdegree are assigned rank $r_v = 1 + n_1$. This accumulation continues such that all nodes in the class with the kth largest outdegree are assigned rank $r_v = 1 + n_1 + n_2 + \ldots + n_{k-1}$. For example, if there are 1000 nodes with outdegree larger than 3, and there are 100 nodes with outdegree 3, then the nodes with outdegree 2 will be ranked 1101. All nodes with same outdegree have the same ranking.

The Eigen exponent power law says that the eigenvalues, λ_i, of a graph are proportional to the order, i, raised to the power of a constant ε. Here, the topology graph is represented as an adjacency matrix and its eigenvectors and eigenvalues are found. The eigenvalues represent the contribution of each eigenvector to the graph, in decreasing order.

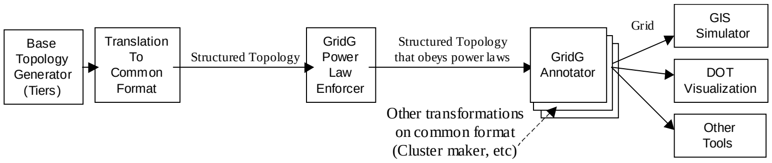

Base Topology Generator (Tiers)
Translation To Common Format
Structured Topology
GridG Power Law Enforcer
Structured Topology that obeys power laws
GridG Annotator
Grid
GIS Simulator
DOT Visualization
Other Tools
Other transformations on common format (Cluster maker, etc)

Figure 1: GridG Architecture.

Symbol	Description	Typical Values or Constraints
R	Rank exponent	≈ -0.488 at router level
O	Outdegree exponent	≈ -2.487 at router level
ε	Eigen exponent	≈ -0.177 at router level
H	Hop-plot exponent	≈ 2.84 at router level
d_v	Outdegree of node v	$1 \leq d_v \leq MaxD$
$MaxD$	Theoretical maximum outdegree	$MaxD = e^{-\frac{\log \alpha}{O}}$
r_v	Ranking of node v	All nodes with same outdegree have same ranking
f_d	Frequency of outdegree d	Number of nodes with outdegree d, $f_d \geq 1$
h	Number of hops	
λ_i	The i^{th} biggest eigenvalue of the graph	
$P(h)$	Total number of pairs of nodes within h hops	
N	Total number of nodes in the graph	
β	Constant in equation 1	$\approx exp(4.395)$ at router level
α	Constant in equation 3	$\approx exp(8.52)$ at router level

Figure 3: Symbols used in this paper.

The hop-plot exponent law is listed as an approximation by Faloutsos, et al. It says that the total number of pairs of nodes, $P(h)$, within h hops, is proportional to the number of hops raised to the power of a constant, H.

3.2 Current graph generators

There are mainly three types of topology generators in use: random [33], hierarchical, and degree-based. Debates as to which type is better for Internet graph generation have persisted over a long period of time [32]. Our belief is that a good graph generator should produce a clear hierarchy that also follows the discovered power laws. Hierarchical generators such as Tiers [7, 14] and Transit-Stub [7] can generate a clear hierarchical network, but the graphs don't follow the power laws by nature. The degree-based generators, such as Inet [21, 34], Brite [24], the CMU power law graph generator [27] and PLRG [2], generate graphs that follow the power laws,but have no clear hierarchical structure. The topologies generated by GridG follow the power laws *and* have a clear three-level hierarchy.

3.3 Algorithms

GridG takes the output of a basic graph generated by Tiers as its input. This input graph has no redundant links. GridG adds links to the graph according to the outdegree power law. Hence, the graphs generated by GridG have a clear three-level hierarchical structure and follow the power laws. The following is a more detailed description.

1 Generate a basic graph without any redundant links using Tiers. Tiers itself has several parameters, specifically the number of nodes and networks at each level of hierarchy. Translate the graph into GridG's native format. This basic graph has three levels of hierarchy: WAN, MAN, and LAN. At each level, the nodes are connected by a minimum spanning tree. Each lower level network is connected to one node on the higher level network. The graph is guaranteed to be connected.

2 Assign each node an outdegree at random using the outdegree power law as the distribution. The probability $P(k)$ that a vertex in the network interacts with k other nodes decays as a power law. This probability is scale-free, meaning that we can extend graphs of any size in this manner [5]. Nodes of outdegree one deviate from the power law as described by Faloutsos, et al [15, Figure 6(b)]. We set $f_1 = f_2$ as this is the case for real router level data. Given outdegree $d = 2, 3 \ldots MaxD$, we calculate the corresponding frequencies according to $f_d = exp(8.9)d_v^{-2.486}$, where 8.9 and -2.486 are the

	Internet Routers	GridG	Tiers
Rank exponent	-0.49	-0.51	-0.18
R^2		0.94	0.89
Outdegree exponent	-2.49	-2.63	-3.4
R^2		0.97	0.55
Eigen exponent	-0.18	-0.24	-0.23
R^2		0.97	0.97
Hop-plot exponent	2.84	2.88	1.64
R^2		0.99	0.99

Figure 4: GridG topology generator evaluation.

defaults for parameters given a configuration file. $N = \sum_{i=1}^{MaxD} f_i$, where N is the total number of nodes in the graph. We then generate a random number x between 1 and N for each node, if $x \leq f_1$, the node is assigned outdegree 1; if $f_1 < x \leq f_1 + f_2$, the node is assigned outdegree 2; if $f_1 + f_2 < x \leq f_1 + f_2 + f_3$, the node is assigned outdegree 3, etc.

3 Calculate the remaining outdegree of each node after taking the links of the minimum spanning tree into consideration.

4 Add redundant links to the graphs by randomly choosing pairs of nodes with remaining outdegree > 0. Nodes at higher levels (e.g., WAN) are given priority over nodes at lower levels (e.g., MAN). Continue to add more redundant links until no pairs of nodes with positive outdegree can be found.

3.4 Evaluation

In this section, we show that the graphs generated by GridG follow the power laws. For comparison, the basic graph generated by Tiers is also shown in our figures. The basic graph was generated by "Tiers 1 50 10 500 40 5 1 1 1 1 1", meaning one WAN containing 50 MANs, each containing 10 LANs. The WAN contains 500 nodes, while the MANs and LANs contain 40 and 5 nodes, respectively. This is similar to the parameters used in other evaluations [32].

Figure 4 shows that the exponents of the topology generated by GridG match router level data from the Faloutsos paper much better than those of the basic Tiers graph. The coefficients of determination, R^2, represent how well a power law fits the generated data. We can see that GridG produces R^2 values close to 1, the ideal. The exponents in Figure 4 are the slopes in Figures 5, 6, 7, and 8.

Figure 5 is a log-log plot of outdegree versus ranking. We can see that a linear fit on this graph explains the relationship for GridG's topology very well. The divergence at small ranks is quite interesting and shows up in studies of real topologies including Faloutsos, et al [15, Figure 4(b)] and Medina, et al [25, Figure 6]. Removing the three diverging datapoints from Figure 5 increases R^2 to 0.99. In Section 4, we will describe a potential new rank law that models this divergence and can be derived from the outdegree law.

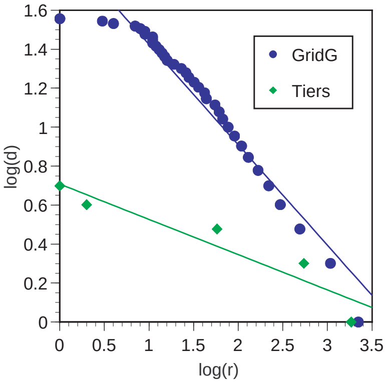


Figure 5: Log-log plot of Outdegree vs. Ranking.

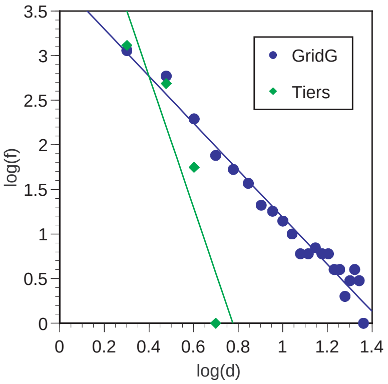


Figure 6: Log-log plot of Frequency vs. Outdegree.

Figure 6 shows a log-log plot of frequency versus outdegree. GridG follows the outdegree exponent power law very well except when outdegree equals 1, which is not plotted in the graphs. We have already noted that this divergence is intentionally induced to better match real topologies.

Figure 7 is a log-log plot of the number of pairs nodes within h hops versus number of hops h. Clearly, GridG's topology conforms to this power law.

Figure 8 is a log-log plot of the eigenvalues in decreasing order. We can see that GridG agrees very well with this power law, though our exponents deviate slightly from

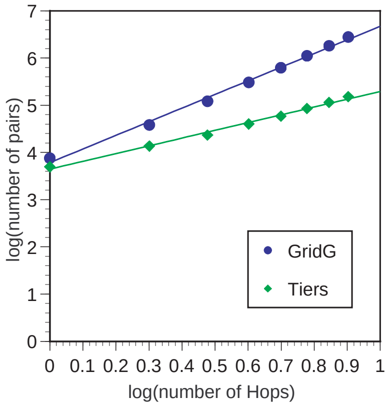


Figure 7: Log-log plot of number of pairs of nodes within h hops vs. number of hops h.

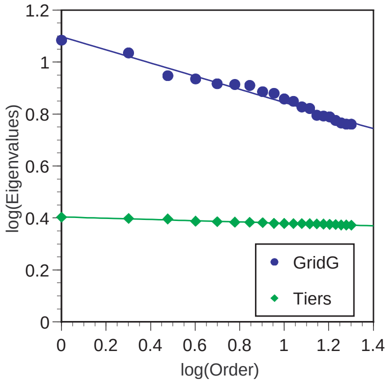


Figure 8: Log-log plot of eigenvalues in decreasing order.

the data given by Faloutsos, et al [15].

4 Relationships among power laws

Several recent graph generators [21, 34, 2, 27] and GridG generate graphs according to the outdegree law only. However, the generated graphs follow all four power laws! Why is this possible? A possible reason is that the power laws (Figure 2) are closely interrelated. A recent paper [5] proposed incremental growth and preferential connectivity to explain the phenomenon and origin of the outdegree law. Medina, et al found that the hop and eigenvalue power laws were followed by all the topologies they considered [25]. Mihail and Papadimitriou have shown that the eigenvalue law follows from the outdegree law [26].

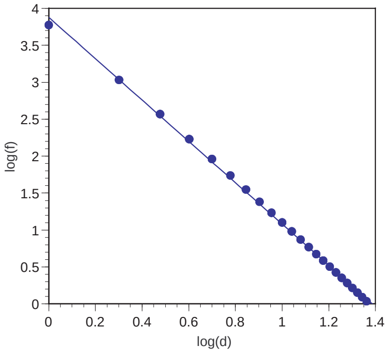


Figure 9: Log-log plot of derived f-d law.

In the following, we show that the outdegree law follows from the rank law. It does not appear, although we can not prove, that the rank law follows from the outdegree law. This suggests one of several possibilities:

- The rank law is strictly more descriptive than the outdegree law.
- The rank law is wrong.
- The outdegree law is wrong.

The evidence against the first and third possibilities is the unreasonable effectiveness of using the outdegree law to generate graphs that appear to follow all of the laws. Furthermore, as we noted earlier, the rank/outdegree relationship diverges from a strict power law in actual topologies at small ranks. Finally, earlier work has shown that the eigenvalue law follows from the outdegree law and that most networks exhibit the eigenvalue and hop-plot laws.

Our belief is that the second possibility is the case. We show that it is possible to derive a power law like rank law from the outdegree law that captures the divergence seen in real topologies and gives the appearance of a power law over much of its range. This also would explain the surprising effectiveness of only using the outdegree law in graph generation. We advocate the following relationship among the laws:

$$\boxed{\text{New rank law} \Longleftrightarrow \text{Outdegree law} \Longrightarrow \text{Eigenvalue law.}}$$

4.1 Rank law $\Longrightarrow$ outdegree law

Starting from the rank law, we derive a form of the outdegree law. Let f_d be the frequency of nodes with outdegree equal to d, or the number of nodes with outdegree d. Let r_d be the ranking of the nodes with outdegree d. Similarly, let r_{d-1} be the ranking of the nodes with outdegree equal to $d-1$. Given the outdegrees d and $d-1$, and the ranking of nodes with those outdegrees, the frequency of outdegree d is

$$f_d = r_{d-1} - r_d \tag{4}$$

Now, substitute for r_{d-1} and r_d their values according to the rank law (Equation 2). This gives

$$f_d = (\frac{d_v - 1}{\beta})^{\frac{1}{R}} - (\frac{d_v}{\beta})^{\frac{1}{R}} \tag{5}$$

To simplify further,

$$\boxed{f_d = \beta^{-\frac{1}{R}}[(d_v - 1)^{\frac{1}{R}} - d_v^{\frac{1}{R}}]} \tag{6}$$

This relationship is itself a power law that associates frequency and outdegree. Figure 9 shows the log-log plot of this derived outdegree law (Equation 6). We have derived an outdegree power law from the rank power law.

4.2 Outdegree law $\Longleftrightarrow$ new rank law

Starting from the outdegree law, we attempted to derive a power law for the rank-outdegree relationship. Our end result is a rank law which is not a power law. If our reasoning is correct, then this shows that the rank law can not be derived from the outdegree law. As discussed earlier, we believe that the new rank law, which we derive from the outdegree law, is more accurate than the original rank law in that it fits actual topology data better.

First, note that

$$\sum_{d=1}^{MaxD} f_d = N \tag{7}$$

where N is the total number of nodes in the graph, $MaxD$ is the maximum outdegree and f_d is the number of nodes with outdegree d, or the frequency of outdegree d. The minimum frequency is 1, so we must make sure that $f_d = \alpha d^O \geq 1$ (Equation 3). Substituting, we see that

$$MaxD = e^{-\frac{\log \alpha}{O}} \tag{8}$$

From the definitions of rank and frequency, we get

$$r_v = 1 + \sum_{d=d_v+1}^{MaxD} f_d \tag{9}$$

That is, the rank of the nodes with outdegree v is equal to one plus the total number of nodes with outdegree bigger than d_v. Using the outdegree law (Equation 3), we get

$$r_v = 1 + \alpha \sum_{d=d_v+1}^{MaxD} d^O = 1 + N - \alpha \sum_{d=1}^{d_v} d^O \tag{10}$$

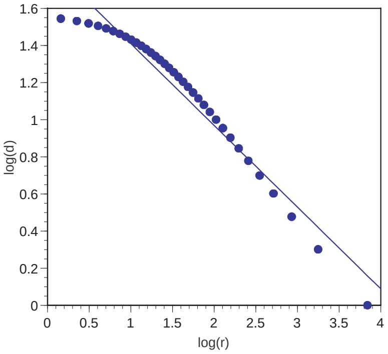


Figure 10: Log-log plot of derived d-r law.

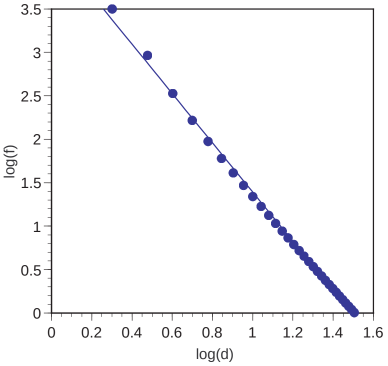


Figure 11: Log-log plot of derived d-f law using the new d-r law.

If we assume that O is a negative rational number, as shown in paper [15], we can then derive the following relationship between rank and outdegree:

$$\boxed{r_v = 1 + N - \alpha[\zeta(-O) - \zeta(-O, 1 + d_v)]} \tag{11}$$

Here, $\zeta(t) = \sum_{n=1}^{\infty} \frac{1}{n^t}$ is the Riemann Zeta function, and $\zeta(-O, 1 + d_v) = \sum_{n=1+d_v}^{\infty} n^O$.

Figure 10 is a log-log plot of this derived rank law. Surprisingly, this derived law is not an ideal power law—it is far from a straight line. If our derivation is correct, it is clear that the rank law does not follow from the outdegree law. Furthermore, our derived law is a better fit to the actual observed topologies than the rank law. A close look at Figure 10 shows that when rank $r >\approx 37$, the relationship between log rank and log outdegree is nearly a straight line, giving the appearance of a power law relationship. The divergence for $r \leq\approx 37$ is similar to that shown for the actual router level topology in Faloutsos, et al [15, Figure 6(b)].

Figure 11 shows the log-log plot of the outdegree and frequency relationship that can be derived from the new rank law (Equation 11) and Equation 4.

5 Annotations

In addition to producing a realistic topology that extends to the level of hosts, a grid generator must also annotate the topology with the attributes of its links, routers, switches, and hosts.

As to the network link bandwidth and latency, we can rely on the output of the underlying structure graph generator, leveraging work in the networking community as discussed earlier. As an alternative, we have also built into GridG an optional feature to explicitly generate network link bandwidth and latency. We assume that these distributions are different at the WAN, MAN and LAN levels, and give the user a mechanism to supply them. Similarly, the distributions of the switching bandwidth of routers are specified by the user in a configuration file. GridG then selects randomly based on the supplied distributions.

Host characteristics are considerably more complex. Our initial approximation is to treat each attribute of a host independently. The user supplies an empirical distribution for the attribute, and we select randomly based on that distribution. Host attributes are not independent, however. This oversimplified approach can generate unreasonable results. Figure 12 shows examples of silly combinations of host configurations that can result. These hosts appear silly because they violate our expectations about how the attributes of an individual machine are likely to be related. For example, we expect that most buyers will scale the memory of a machine with the number of CPUs, and that software vendors rarely sell their competitor's OS.

There also exist correlations between the attributes of machines that are near each other in the network topology. The obvious example is a cluster, in which tightly coupled machines have identical attributes. While we can support clusters via an additional graph transformation that explicitly creates them, there are other examples: servers in the same machine room are likely to have more in common with each other than with the client hosts that use them.

To capture such intra-host and inter-host attribute correlations, we extended GridG's annotator with a general engine that supports user-supplied conditional probability rules. For example, the user can assert that hosts with four or more processors have at least 4 GB of memory. In the following, we describe the engine, a set of core "common sense" intra-host rules used to prevent silly hosts, an inter-host OS concentration rule for subnets derived from measurement, and show examples of sensible annotations. More rules are necessary and need to be derived from measurement. We describe our efforts to capture sufficient measurements to do so.

5.1 Annotation algorithm

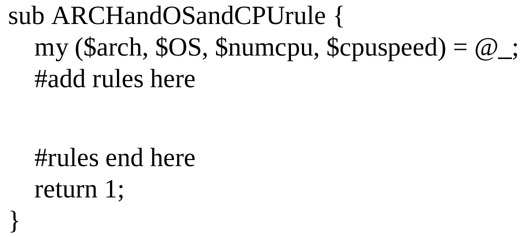
Given a model of the distributions and correlations of host characteristics, it would be relatively straightforward to generate realistic host attribute information. However, at the present time we do not have such a complete model because of the difficulties in collecting a sufficient amount of data from which to infer the model. We appear to be the first to need to do this, and we have tried a number of techniques to acquire data, which we discuss later.

In light of the limited data, we have made GridG annotations conform to user-supplied rules and created a set of common-sense rules. In this way, the current generated host attribute information is reasonable on its face, and as new distributions and correlations are discovered, the user can add rules to GridG to make the generated grids conform. In the current implementation, user rules take the form of Perl functions. *Frames* for those functions are given to which the user can simply add their rules. For example, Figure 13 shows a function frame governing the correlation among CPU architecture, OS, the number of CPUs and CPU clock rate. Figure 14 shows two example rules added to this frame. The first rule says that Intel 32 and 64 bit architecture machines support no more than 4 CPUs, while the second rule says that for any architecture, the maximum number of CPUs per host is 1024. Figure 15 presents GridG's default rules in English. These rules can be removed or enhanced by the user and new rules can be added.

We believe that there are at least two types of correlations among the host's attributes. The first type is the correlations among the different attributes of an individual host, namely, correlations among the number of CPUs, CPU clock rate, memory size, disk size, etc. We can easily imagine important correlations of this type. For example, we assume a positive correlation between the number of CPUs and the total memory, and that machines with more CPUs are less likely to run a version of Windows. These assumptions appear in our default rules as shown in Figure 15. Some of those correlations are deterministic (e.g.,

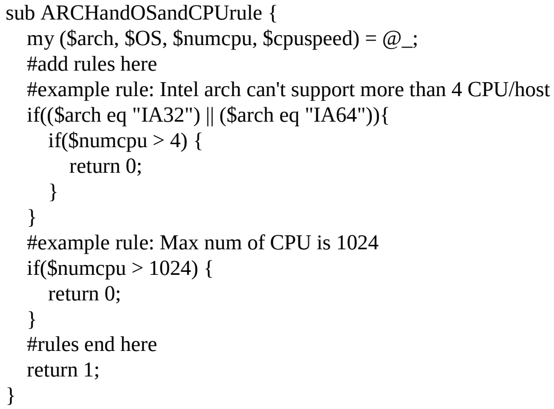
Host	CPUS	CPU MHZ	Memory (MB)	Disk (GB)	Arch	OS	OS vendor	Current Load
1	512	1200	256	40	IA32	DUX	Sun	0
2	16	1000	512	800	PARISC	NetBSD	Microsoft	3
3	4	1600	512	160	SPARC32	DUX	RedHat	1
4	1	1800	65536	400	IA32	Solaris	Microsoft	2

Figure 12: Silly host configurations generated by the initial GridG annotator.



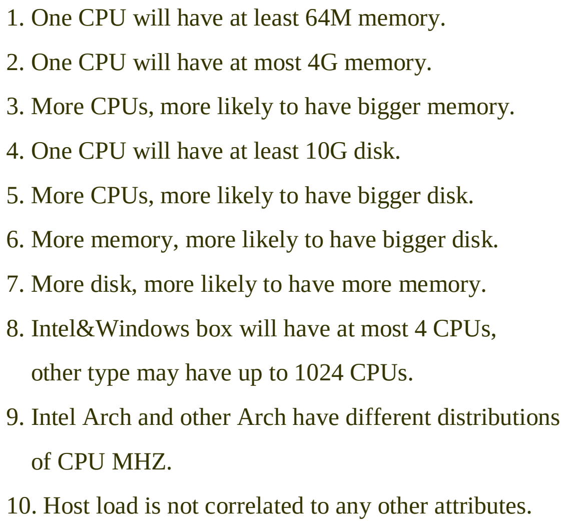
```
sub ARCHandOSandCPUrule {
   my ($arch, $OS, $numcpu, $cpuspeed) = @_;
   #add rules here


   #rules end here
   return 1;
}
```

Figure 13: Rule frame governing the correlation among CPU architecture, OS, number of CPUs, and CPU clock rate.



```
sub ARCHandOSandCPUrule {
   my ($arch, $OS, $numcpu, $cpuspeed) = @_;
   #add rules here
   #example rule: Intel arch can't support more than 4 CPU/host
   if(($arch eq "IA32") || ($arch eq "IA64")){
      if($numcpu > 4) {
         return 0;
      }
   }
   #example rule: Max num of CPU is 1024
   if($numcpu > 1024) {
      return 0;
   }
   #rules end here
   return 1;
}
```

Figure 14: Example rules.



1. One CPU will have at least 64M memory.
2. One CPU will have at most 4G memory.
3. More CPUs, more likely to have bigger memory.
4. One CPU will have at least 10G disk.
5. More CPUs, more likely to have bigger disk.
6. More memory, more likely to have bigger disk.
7. More disk, more likely to have more memory.
8. Intel&Windows box will have at most 4 CPUs,
   other type may have up to 1024 CPUs.
9. Intel Arch and other Arch have different distributions
   of CPU MHZ.
10. Host load is not correlated to any other attributes.

Figure 15: Default rules.

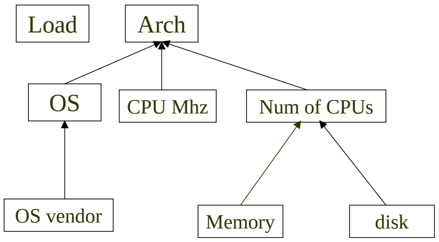


Figure 16: Dependence tree.

a machine with 16 CPUs can't run windows as its operating system), and others are probabilistic (e.g., a machine with 32 CPUs is likely to have more memory per CPU than a 2 CPU machine, but not necessarily.)

The second type is a correlation between the attributes of machines that are near each other in the network topology, namely, correlations such as OS concentration on IP subnets as described in Section 5.2 and shown in Figure 23. This type of correlation is mostly probabilistic.

To make the GridG annotator conform to the first type of correlations, we assume a dependence tree as shown in Figure 16. Host load is independent of other attributes in our model. Architecture is the root of the tree on the assumption that it is the most significant attribute, and that it largely decides what OSes can run on a host and how many CPUs it is likely to have. Clearly the OS vendor depends on the OS. We believe that memory and disk size is likely to grow with the number of CPUs on a machine.

With this dependence tree, we can make GridG conform to correlations by applying conditional probability, choosing the distribution of an attribute based on at-

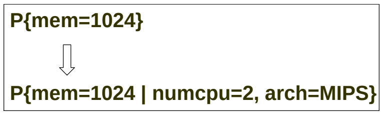


Figure 17: Example of conditional probability.

tributes picked before it. For example, we would first pick the architecture, then the number of CPUs based on that choice, and finally the amount of memory based on those two choices, as shown in Figure 17. This approach appears to work well in practice, generating sensible hosts. In the following, we present in detail the algorithm used to annotate the hosts on a LAN. This process is repeated for each LAN in the topology. To make the description easier to understand, we present the flow chart of the algorithm in Figure 18.

1. If the OS concentration feature (Section 5.2) is turned on, then, for each IP subnet (generated LAN in our topology) , select the OS concentration percentage P from the user configuration file. Next, select the dominant architecture for the subnet according to the user configuration file. Select the dominant OS according to the dominant architecture for the LAN. There is one dominant architecture and OS for each LAN.

2. For each host, generate architecture for the host according to distribution specified in the user configurable file. If the OS concentration feature is turned on, change the host architecture to the dominant architecture with probability P.

3. Architecture is subdivided into several groups, such as the Intel Architecture family, MIPS, etc. Each group has its own specified distribution for OS, number of CPUs and CPU clock rate in the configure file. Using the distributions, generate the OS according to the architecture type. If the OS concentration feature is turned on and the architecture has been changed to the dominant architecture, also change the generated OS to the dominant OS.

4. Using the distributions, generate CPU clock rate and the number of CPUs according to the architecture type.

5. Apply the user rule governing the correlation among architecture, OS, number of CPUs and CPU clock rate. If the rule is not satisfied, go back to the last step.

6. The number of CPUs is subdivided into several groups, such as the number of CPUs, $n \leq 4$, $4 < n \leq 32$, $32 < n \leq 128$, etc. Each group has its own memory and disk size distribution specified in the configuration file. The configuration is such that a group with a large number of CPUs has distributions for memory and disk size that are of higher mean than those for a group with a small number of CPUs. Within each group, we apply a *promotion probability function* and and a *promotion rate function* so that a machine with larger number of CPUs is more likely to increase its memory and disk. Optionally, we apply a *degradation probability function* and *degradation rate function* so that a machine with smaller number of CPUs is more likely to decrease its memory and disk, which is a strengthening mechanism for promotion probability and rate functions. We discuss these functions in more detail below.

7. Apply the user rule governing the correlation among the number of CPUs, memory size, disk size, and OS, architecture, and CPU clock rate. If the correlations are not satisfied, go back to last step.

8. Generate OS vendor according to its OS.

9. Apply the user rules governing the correlation between OS and OS vendor. If the correlation is not satisfied, go back to last step.

10. Generate a load value for the host according to the specified distribution in user configuration file.

11. Apply the user rule governing the correlation between load and other attributes. If it is not satisfied, go back to last step.

12. Apply the overall user rule governing architecture, OS, number of CPUs, CPU clock rate, memory size, disk size, OS vendor and load of the host, if it is not satisfied, go back to step 4.

13. If all the hosts on the LAN are annotated, go to next step, otherwise, go to step 2.

14. Terminate.

The promotion probability function and the degradation probability function map from the number of CPUs to a probability PP, the probability that memory and disk size will be increased (promotion) or decreased (degradation) based on a host's number of processors. A larger

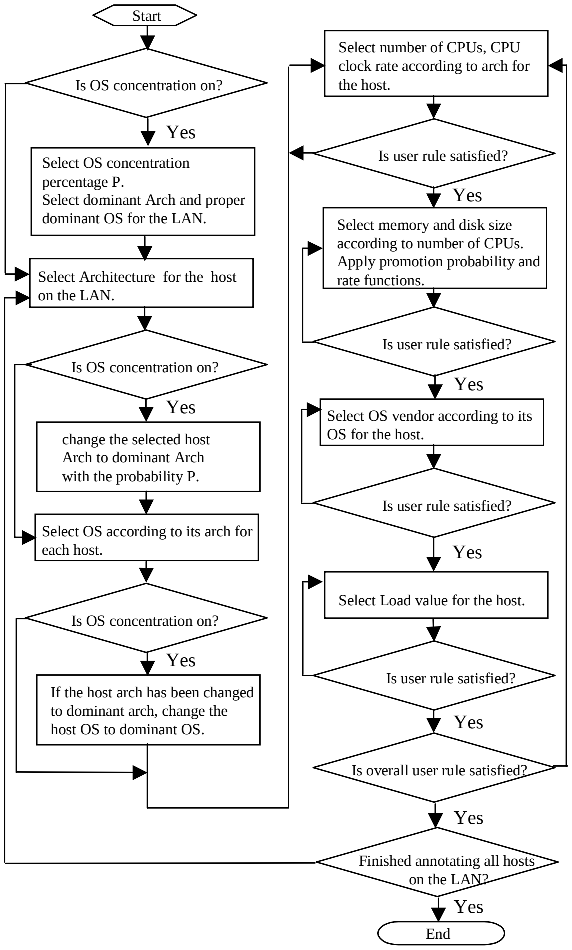

Start
Is OS concentration on?
Yes
Select OS concentration percentage P.
Select dominant Arch and proper dominant OS for the LAN.
Select Architecture for the host on the LAN.
Is OS concentration on?
Yes
change the selected host Arch to dominant Arch with the probability P.
Select OS according to its arch for each host.
Is OS concentration on?
Yes
If the host arch has been changed to dominant arch, change the host OS to dominant OS.
Select number of CPUs, CPU clock rate according to arch for the host.
Is user rule satisfied?
Yes
Select memory and disk size according to number of CPUs. Apply promotion probability and rate functions.
Is user rule satisfied?
Yes
Select OS vendor according to its OS for the host.
Is user rule satisfied?
Yes
Select Load value for the host.
Is user rule satisfied?
Yes
Is overall user rule satisfied?
Yes
Finished annotating all hosts on the LAN?
Yes
End

Figure 18: Flow chart of the algorithm.

```
sub promotionProb
{
    my($numcpu, $maxcpu, $mincpu) = @_;
    my $pp100;
    $pp100=100*($numcpu-$mincpu)/($maxcpu-$mincpu);
    return $pp100;
}
```

Figure 19: Linear promotion probability function.

number of processors leads to an increased promotion and a decreased degradation probability. Paired with these are the promotion and degradation rate functions, which determine the extent to which memory and disk size will be changed. The default rate function doubles/halves the memory and disk size, but user can specify their own rate and promotion/degradation functions. Figure 19 shows a linear promotion probability function, while Figure 20 shows a power function where the probability of promotion increases faster with the number of CPUs.

```
sub promotionProb
{
    my($numcpu, $maxcpu, $mincpu) = @_;
    my $pp100;
    $pp100=100*sqrt(($numcpu-$mincpu)/($maxcpu-$mincpu));
    return $pp100;
}
```

Figure 20: Power promotion probability function.

Promotion function	Correlation coefficient
None	0.69
Linear	0.73
Power	0.76

Figure 21: Influence of promotion probability and rate functions on correlation coefficient between number of CPUs and memory or disk.

Together, the promotion/degradation probability and rate functions control the correlation coefficient between number of CPUs and memory or disk sizes. Shown in Figure 21 is the influence of promotion probability and rate functions. If stronger correlations are required, both promotion and degradation probability and rate functions are chosen to increase faster with the number of processors.

This algorithm and the default rules work well and generate results that are sensible. More rules can be added as they are discovered. Figure 22 shows a few reasonable hosts generated by the current GridG implementation.

5.2 OS concentration rule

The nmap port-scanning tool [19] has the ability to determine a host's operating system based on how its TCP/IP implementation behaves. We used nmap to scan 10 different class C IP networks, both at Northwestern and belonging to a company that maintains a popular web site. In each subnet we oberved OS concentration to various degrees, that is, there was typically one dominant operating system.

Host	CPUS	CPU MHZ	Memory (MB)	Disk (GB)	Arch	OS	OS vendor	Current Load
1	512	1200	65536	10240	MIPS	FreeBSD	FreeBSD	9
2	16	1000	8192	800	PARISC	NetBSD	NetBSD	4
3	4	1600	1024	160	SPARC32	Solaris	Sun	1
4	1	1800	512	80	IA32	Win2k	Microsoft	3

Figure 22: Sensible hosts generated by current GridG annotator.

Organization	Subnet	Dominant OS	Percentage of concentration	Percentage of recognized OS
CS Department	1	Windows	79/95 = 83.2%	78%
	2	Windows	31/37 = 83.8%	81%
	3	Linux	40/40 = 100%	100%
ECE Department	4	Solaris	67/76 = 88.2%	65%
	5	Solaris	41/41 = 100%	94%
	6	Solaris	21/21 = 100%	96%
A popular web site	7	FreeBSD	214/214 = 100%	87%
	8	FreeBSD	187/187 = 100%	89%
	9	FreeBSD	191/192 = 99%	92%
	10	FreeBSD	72/73 = 99%	91%
Total machines		976		

Figure 23: OS concentration observed in IP subnets.

Figure 23 shows the data for all the 10 subnets. Nmap couldn't recognize every host's OS from its TCP/IP fingerprints because nmap doesn't have fingerprints for all versions of every OS. However, notice that even if we assume that all the unrecognized OSes were of a type other than the dominant OS, we can still assert that OS concentration exists in all 10 subnets. Subnet 3 is known to be a Linux cluster but there is no cluster in subnet 4, 5 and 6.

People are very sensitive to the port scanning and therefore we have had to limit our activity. However, given this sample of 976 machines and 10 subnets, it seems likely that such OS concentrations occur on many subnets. It also appears sensible given that subnets are owned by individual organizations, and many organizations have standardized operating systems and even machines. Another factor is the existence of clusters in which every node has identical hardware and software. We have added this OS concentration behavior to the GridG rulebase as an optional feature.

5.3 Measurements for more rules

To add more rules to GridG, we need a large random sample of hosts on the Internet or from a representative grid environment. For each host, we need information about its hardware and software resources, and a description of where it is on the network topology captured in the sample. We have considered the following ways of acquiring such a sample:

- Examine the contents of existing grid information service systems. We studied the (anonymized) dumps from several large grids and the dataset collected by Smith, et al [30], which was collected when there was a single MDS server. Smith's dataset was by far the largest, but it contained fewer than 1000 machines. Our conclusion is that existing systems don't contain enough data for our needs.
- Use SNMP scans. The default SNMP MIB provides almost all the information we need. However, most users have turned off SNMP on their hosts due to security concerns. This results in a small and biased sample.
- Use DMI/IPMI/OpenManage/etc. These are BIOS-level distributed management tools for PCs. The prospects here are not clear yet, but even if they could be used, they would provide a sample biased towards PCs.
- Write a virus. An innocuous virus that reported back host data and deleted itself after infecting several other machines would be highly effective, although the sample would be biased towards machines exhibiting the exploits used. We discarded this idea because of its ethical implications.
- Use hardware vendor sales data. If we knew of new machines being sold and attached to the net-

work, we could at least derive distributions of their attributes. We attempted to acquire sales data from IBM, Dell, and HP, but were unsuccessful. These companies regard even aggregated sales data as proprietary information.

- Use OS vendor registration data. Hooking into the processes of OS registration would provide very detailed, if OS-biased information. We attempted to establish a relationship with Microsoft and with Red Hat, but were unable to do so.

- Use open source peer-to-peer software such as Gnutella. Gnutella has a large number of users, so we might expect that we could use it to acquire information about the host machines. Unfortunately, Gnutella doesn't expose any machine configuration data that would be of use to us.

- Continue to use network exploration tools such as nmap. While this was very successful in finding the OS concentration property, it is unclear how much more data it can produce. Also unfortunate, running such tools to explore a remote network is often perceived as a hostile act.

We continue to explore new ways to collect data to expand our rulebase.

6 Conclusion

We have presented GridG, a tool for synthesizing realistic computational grids. GridG produces structured network topologies that obey the power laws of Internet topology. While developing GridG's topology generator, we found that two of the power laws (rank and outdegree exponent laws) are in conflict. We derived a new rank law from the outdegree law that conforms well with published data on actual topologies and has a power law like range. We speculate that this new law is a better approximation of rank behavior.

The topology annotation component of GridG can annotate the network according to user supplied empirical distributations and user conditional probability rules. Two kinds of correlations among hosts attributes are considered and built into GridG: correlations between an individual host's attributes and correlations between attributes on nearby hosts. We developed a basic set of rules that capture common sense intra-host correlations. Through nmap based measurement, we observed OS concentrations in all the IP subnets we probed. We added this as an inter-host rule.

Developing additional rules will require more host measurement data to analyze. We have tried a number of techniques for acquiring such data and have been largely unsuccessful. We continue to explore ways to acquire a rich set of measurements from which to derive a larger rulebase for GridG annotations.

GridG version 1.0 is available online at the following URL: http://www.cs.northwestern.edu/∽urgis/GridG.

Acknowledgments

We would like to thank Dr. Huaien Li and Mr. Yi Qiao for discussions on this paper.

References

[1] AIDA, K., TAKEFUSA, A., NAKADA, H., MATSUOKA, S., SEKIGUCHI, S., AND NAGASHIMA, U. Performance evaluation model for scheduling in a global computing system. *International Journal of High Performance Computing Applications 14*, 3 (2000).

[2] AIELLO, W., CHUNG, F., AND LU, L. A random graph model for massive graphs. In *ACM Symposium on Theory of Computing* (2000).

[3] ALBERT, R., AND LASZLO BARABASI, A. Statistical mechanics of complex networks. *Reviews of modern physics 74* (2002).

[4] BANERJEE, S., LEE, S., BHATTACHARJEE, B., AND SRINIVASAN, A. Resilient multicast overlays. In *Sigmetrics 2003* (2003).

[5] BARABASI, A., AND R.ALBERT. Emergence of scaling in random networks. *Science* (1999), 509–512.

[6] BUYYA, R., AND MURSHED, M. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)* (May 2002 (to appear)).

[7] CALVERT, K. L., DOAR, M. B., AND ZEGURA, E. W. Modeling internet topology. *IEEE Communications Magazine 35*, 6 (June 1997), 160–168.

[8] CASANOVA, H. Simgrid: A toolkit for the simulation of application scheduling. In *the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)* (2001).

[9] CZAJKOWSKI, K., FITZGERALD, S., FOSTER, I., AND KESSELMAN, C. Grid information services for distributed re-

source sharing. In *10th IEEE Symp. on High Performance Distributed Computing* (2001).

[10] DINDA, P., GROSS, T., KARRER, R., LOWEKAMP, B., MILLER, N., STEENKISTE, P., AND SUTHERLAND, D. The architecture of the remos system. In *10th IEEE Symp. on High Performance Distributed Computing* (2001).

[11] DINDA, P., AND PLALE, B. A unified relational approach to grid information services. *Grid Forum Informational Draft GWD-GIS-012-1* (February 2001).

[12] DINDA, P. A. Online prediction of the running time of tasks. *Cluster Computing 5*, 3 (2002).

[13] DINDA, P. A., AND LU, D. Nondeterministic queries in a relational grid information service. In *Proceedings of ACM/IEEE SC 2003 (Supercomputing 2003)* (2003). To Appear. (In this volume.).

[14] DOAR, M. B. A better model for generating test networks. *IEEE GLOBECOM* (1996).

[15] FALOUTSOS, M., FALOUTSOS, P., AND FALOUTSOS, C. On power-law relationships of the internet topology. In *Proceedings of SIGCOMM '99* (1999).

[16] FOSTER, I. The anatomy of the grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science 2150* (2001), 1–25.

[17] FOSTER, I., AND KESSELMAN, C. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing 11*, 2 (Summer 1997), 115–128.

[18] FOSTER, I., AND KESSELMAN, C., Eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[19] FYODOR. Remote os detection via tcp/ip stack fingerprinting. (web page). http://www.insecure.org/nmap/nmap-fingerprinting-article.html.

[20] GRIMSHAW, A. S., WULF, W. A., AND TEAM, C. T. L. The legion vision of a worldwide virtual computer. *Communications of the ACM 40* (1997), 39–45.

[21] JIN, C., CHEN, Q., AND JAMIN, S. Inet: Internet topology generator. Tech. Rep. CSE-TR443-00, Department of EECS, University of Michigan Ann Arbor, 2000.

[22] KLEINBERG, J. M., KUMAR, R., RAGHAVAN, P., RAJAGOPALAN, S., AND TOMKINS, A. S. The web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science 1627* (1999), 1–18.

[23] LOWEKAMP, B. B. Combining active and passive network measurements to build scalable monitoring systems on the grid. *Performance Evaluation Review 30*, 4 (March 2003), 19–26.

[24] MEDINA, A., LAKHINA, A., MATTA, I., FAMEDINA, J. B., AND ANUKOOL. Brite: An approach to universal topology generation. In *IEEE MASCOTS '01 (Tools track)* (2001).

[25] MEDINA, A., MATTA, I., AND BYERS, J. On the origin of power laws in internet topologies. *ACM SIGCOMM Computer Communication Review (CCR) 30* (2000), 18–28.

[26] MIHAIL, M., AND PAPADIMITRIOU, C. On the eigenvalue power law. *Springer-Verlag Lecture Notes in Computer Science* (2002).

[27] PALMER, C. R., AND STEFFAN, J. G. Generating network topologies that obey power laws. In *GLOBECOM '2000* (2000).

[28] RIPEANU, M., AND FOSTER, I. Mapping gnutella network: Macroscopic propersies of large-scale peer-to-peer systems. In *1st International Workshop on Peer-to-Peer Systems* (2002).

[29] ROWSTRON, A. I. T., AND DRUSCHEL, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Middleware* (2001), 329–350.

[30] SMITH, W., WAHEED, A., MEYERS, D., AND YAN, J. C. An evaluation of alternative designs for a grid information service. *Cluster Computing 4* (2001), 29–37.

[31] SONG, H. J., LIU, X., JAKOBSEN, D., BHAGWAN, R., ZHANG, X., TAURA, K., AND CHIEN, A. A. The microgrid: a scientific tool for modeling computational grids. In *Supercomputing* (2000).

[32] TANGMUNARUNKIT, H., GOVINDAN, R., AND JAMIN, S. Network topology generators: degree-based vs. structural. In *Proceedings of SIGCOMM '02* (2002).

[33] WAXMAN, B. Routing of multipoint connections. *IEEE J. of Selected Areas in Communications 6*, 9 (1988), 1622–1671.

[34] WINICK, J., AND JAMIN, S. Inet-3.0: Internet topology generator. Tech. Rep. CSE-TR-456-02, Department of EECS, University of Michigan Ann Arbor, 2002.

[35] WOLSKI, R., SPRING, N. T., AND HAYES, J. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems 15*, 5–6 (1999), 757–768.