GridRod – A Dynamic Runtime Scheduler for Grid Workflows

Shahaan Ayyub
Monash University
MESSAGE LAB, Faculty of Information Technology,
Monash University Australia, 3145
+61 (03) 9903 1217
ayyub.shahaan@infotech.monash.edu.au

David Abramson
Monash University
MESSAGE LAB, Faculty of Information Technology,
Monash University Australia, 3145
+61 (03) 9905 1183
davida@infotech.monash.edu.au

ABSTRACT

Grid Workflows are emerging as practical programming models for solving large e-scientific problems on the Grid. However, it is typically assumed that the workflow components either read or write data to conventional files, which are copied from one execution stage to another, or they are tightly coupled using IPC libraries such as MPI or distributed streaming. More flexible communication can be achieved by overloading conventional READ and WRITE operations with advanced IO mechanisms such as sockets, streams and pipes, as is done in the GriddLeS environment. Such flexibility allows the pipelining of temporally dependent components, or in contrast, delaying of tightly coupled computations based on the current resource availability and network connectivity. However, it is also harder to schedule the workflow, because the communication mode may not be decided until run time. In this paper, we propose a new scheduling model that leverages such communication flexibility and allows us to generate dynamic runtime schedules. The scheduler in this case, not only allocates components to distributed Grid resources, but also specifies the inter-component communication mechanism (socket, pipe etc.) The current model is implemented as a dynamic workflow scheduling tool called **GridRod**, which harnesses Nimrod/G's [1] Grid services and GriddLeS [2] web services.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Heuristic Methods and Scheduling – *runtime resource allocation and communication specification, spatio/temporal concurrency*.

General Terms

Algorithms, Management, Performance, Design, Reliability.

Keywords

Models of Computation, runtime scheduling, communication specification, spatial and temporal concurrency.

1. INTRODUCTION

The Grid Virtual Organization (VO) model integrates distributed resources such as high performance clusters, distributed data repositories, scientific instruments and specialized hardware devices to provide a collaborative platform to perform e-Science [3, 4]. Such platforms are used to solve large-scale problems by deploying complex application models, such as GENIE [1], specified as Grid workflows. The components of these workflows may consist of monolithic applications, data processing software, scientific instruments, visualization interfaces, or even remote Grid/Web services. Conceptually, formulation of this single "integrated application model" is fairly simple. However, its deployment and scheduling on highly distributed, dynamic and heterogeneous Grid environments is both interesting and challenging.

Accordingly, several projects such as ICENI [5], GridRPC [6] and VGrADS [2] implement a range of scheduling models for achieving high application performance whilst meeting such challenges. However, existing work does not allow the workflow to adapt to the underlying resource base that is available at execution time. Typically, there are two orthogonal approaches for modeling Grid workflows. The first approach temporally orders the workflow execution, where data is transferred from one execution stage to another as files. The second approach binds the components as co-executing computations communicating through unidirectional FIFO channels via pipes, streams or message passing. Both approaches tightly couple the communication and thereby restrict the runtime component allocation. Importantly, a designer specifies which approach to use statically when the workflow is built.

Communication libraries such as GriddLeS [2], allows the communication mode to be delayed until run time. This means that it is possible to choose the most appropriate mechanism depending on the resource base. For example, if there are sufficient resources, then it may be possible to co-schedule a number of the workflow components, and have these run concurrently. On the other hand, if there are not enough resources to run the components at the same time, then it is best to run each one sequentially, and to write results to intermediate files in between. GriddLeS provides this flexibility by intercepting and redirecting primitive IO operations (such as READ and WRITE) to a local file, a remote file or a remote socket. In this way, application components behave as if they are executing in a conventional file system whilst leveraging the distributed computational power of the Grid. Such flexible specification allows pipelining temporally dependent components or, in contrast, delaying tightly coupled co-executing computations. Furthermore, this exposes both temporal and spatial concurrency in the Grid workflows, which can be exploited for achieving high throughput. Dynamic specification of inter-component communication in this way provides more opportunities to optimize runtime component allocation based on the current Grid state.

In this paper, we propose a new model, which makes flexible runtime decisions for scheduling Grid workflow components and for specifying inter-component communication behaviour. The model leverages the IO mechanisms already provided by GriddLeS to generate lazy runtime schedules. In a lazy approach the runtime scheduling decisions are delayed as much as possible. This allows the scheduling model to use the current information about the resource availability and network connectivity for achieving the optimisation. So, in cases where the data generation and consumption is continuous, the scheduler pipelines the distributed components when sufficient computational and network resources are available. In contrast, if the application components have producer-consumer relationships, then the model delays the execution of the downstream computation in case of resource unavailability or in order to optimise the execution overlap. So, if the data transfer on IO channels is unidirectional, the data can be temporarily stored in a buffer at the writer's end and later copied to the reader's location for its execution when sufficient resources become available. This would remove the necessity to execute the tightly coupled applications together, providing opportunities to optimise scheduling as well as improving resource utilization and consequently, applications performance. The scheduling function in this case is therefore a composite function, which not only allocates computational components to Grid resources, but also specifies how they should interact. The proposed model is embedded in the Nimrod/G framework [1], leveraging the flexible IO infrastructure already provided by GriddLeS. This infrastructure provides opportunities for spatial (parallelism) as well as temporal (pipelining) co-execution of the components. In order to maximize the throughput, both types of concurrency should be exploited. However, most Grid scheduling heuristics [2, 5, 7] [8, 9] tend to search in only one of the two orthogonal directions, exploiting only one type of concurrency. To address this, we propose a new class of scheduling heuristics called HyBD, which represents the hybrid of Breadth First and Depth First. The heuristics iteratively explores the workflow graph in both directions to optimise scheduling. Two novel heuristics, HyBD_MAKESPAN and HyBD_DELAY, are proposed under this class with different objective functions. The resulting infrastructure called GridRod is a step towards building a service-oriented architecture (SOA) for Grid Workflow Orchestration. GridRod leverages GriddLeS web services and Nimrod/G's Grid services to achieve the same.

2. Grid Workflows; An orchestration perspective

This section discusses Grid Workflows and workflow modeling techniques mainly from the orchestration and scheduling perspective. Synchronous Data Flow (SDF) [10] and Kahn process Nets (PN) [11], the two most prevalent workflow-modeling approaches, are described. These models are inherited from the set of existing models of computation in Digital Signal Processing (DSP). Variants of these models have been used in the projects such as Kepler [7] and ICENI for specifying Grid workflows, with some nomenclature differences. In the rest of the paper, we use the Kepler nomenclature (PN and SDF) as a reference.

General definition: We define Grid workflows as an integration of distributed standalone components interacting with each other by exchanging data through flexible communication links. The workflow is initially defined in an abstract form followed by the concrete mapping or triggering of components by an orchestration engine to appropriate Grid resources, whilst respecting dependencies. Also, Grid workflows can be represented as Directed Acyclic Graphs (DAGs). In this perspective, workflow modeling defines the workflow components as DAG nodes and their relationships as edges.

2.1 Grid Workflow Models

- *SDF* – Components in Grid-based SDF models are interconnected as temporally dependent units. Each component reads and writes data to a local file, in which case the inter-component communication is established by simply copying the data generated by an upstream writer to the reader's location. Currently, SDF is the most common model for Grid workflow specification as it allows integration of isolated components using a simple reader/writer relationship. However, for the very same reason, pipelining between these components becomes challenging.
- *PN* – Components in a PN model are linked by unidirectional first-in-first-out (FIFO) channels. All components communicate through these channels and their execution is synchronized (by essentially blocking and unblocking the reader process) depending on the data availability on the input channels. Examples of Grid based process networks include distributed streaming applications such as *count-Samps* [12] and *Clust-streams* [13]. These applications occur in scientific areas where a large amount of data is being continuously generated and must be processed in real-time. The Gates project [14], specifically targets scheduling and migration of these applications. However, it is assumed that applications modeled under PN have a communication layer already present between the components for coherent inter-component communication.

2.2 Static Modeling

Current Grid workflow modeling approaches tightly couple the inter-component communication, restricting the lazy runtime scheduling based on the latest CPU and network state. So, if the workflow is modeled as an SDF or a PN, then the components should communicate according to the tight coupling entailed by the model. This limits their dynamic allocation to distributed resources. A Grid workflow may well be constituted from a variety of components and therefore may have its sections modeled differently (as SDF or PN).

As mentioned earlier, if primitive I/Os such as read () and write () can be overridden by sockets, streams and pipes, then temporally dependent computations (modeled as SDF) can be pipelined provided the data is written and read continuously and sufficient resources are available. In this way SDF applications can behave as a network of dataflow processes, which is a special case of PN, and the main focus of our current work.

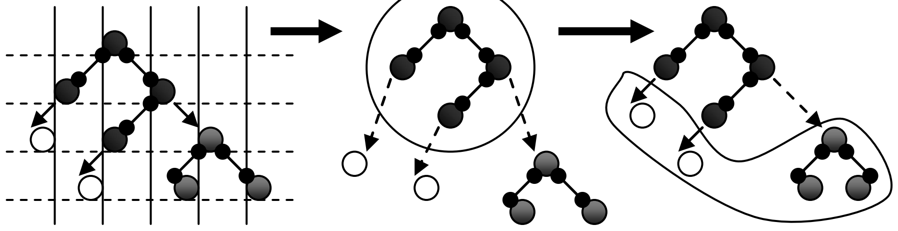


Figure 1. The graph has symmetric depth and breadth. The breadth of a graph defines the spatial concurrency between the components. The different shades of Gray represent two different clusters of PN within the same graph. The round-ended arrows mean that the two connected components can be streamed and concurrently executed. The large arrows show different scheduling states.

PN models, while providing concurrency and parallelism, tightly couple the communication between application components, which are assumed to interact using a pre-existing communication mechanism. Further, co-scheduling workflow components might not be optimal at all times. Consider a situation where a downstream reader takes less time to execute than its parent writer, or where the upstream writer takes only half of its execution period to generate the data that is to be consumed by the reader. Co-scheduling the components in these situations (a typical static co-scheduling approach [15]) would hold the reader's resource while waiting for the data availability, leading to poor resource utilization. Further, it is very likely, in highly dynamic environments, that by the time the reader is actually ready for execution, its resource is no longer the 'best resource' or even available for the execution. In these scenarios, a better approach would be to delay the execution of the downstream reader.

The knowledge of inherent communication behavior of the components is important in optimizing the scheduling. Also, such knowledge can be utilized to understand the bandwidth/latency and data requirements of the components thereby providing more chances for the scheduler to make optimized runtime decisions.

In the next section we describe our scheduling model, which dynamically schedules workflow components as clusters of PN and SDF based on their communication behavior. The scheduling model leverages the IO infrastructure provided by GriddLeS to achieve such modeling. Applications exhibiting asynchronous read/write patterns specifically benefit from such scheduling.

3. Dynamic Scheduling Model

Our model adopts a two-step process to optimize runtime scheduling of the workflow components. The first step is the clustering, which involves analytical selection of downstream components based on their communication behavior. The second step involves the following:

- Allocation of the selected components to Grid resources based on data availability and CPU performance.
- Selection of the appropriate communication mechanism to achieve better runtime application performance.

A new heuristic model called HyBD is proposed in this paper and two novel heuristics based on the proposed model are described.

3.1 HyBD Heuristics

HyBD stands for **Hy**brid of **B**readth First and **D**epth First, and signifies a hybrid of the two prevalent search approaches. A problem space can be described as a Directed Acyclic Graph (DAG) in at least two orthogonal representations, namely spatial and temporal [16]. In a spatial representation, all the components exist simultaneously and are executed concurrently, (spatial parallelism). On the other hand, in a temporal representation, the components have temporal precedence and are executed either in a sequential or a concurrent pipeline, depending on their communication behavior. The key to improve throughput is to explore both types of concurrency in order to optimize scheduling. However, existing graph exploration algorithms search in *either* of the two orthogonal directions (spatial or temporal), rather than in both. Thus an algorithm such as breadth first would explore the DAG spatially, whereas, the depth first would search it temporally. On the other hand, we are interested in finding a local optimization and therefore intend to explore the graph in both directions. For this, we have developed variants of Breadth and Depth First search approaches, which performs repeated searches along the orthogonal directions until a halting condition is encountered. In which case, each repetition essentially appends the visited nodes to a node-cluster list. However, the ordering in which the nodes are visited depends on the search method and the depth and breadth limits are obtained dynamically. This approach provides us with local search completeness to optimize component clustering.

3.2 Component Clustering

The clustering depends on the data generation and consumption patterns of the workflow components. Oldfield and Koltz [17], outline several scientific applications and their I/O behavior, including medical applications, seismic imaging, climate modeling, computational chemistry and biology. The report implies that most applications show asynchronous I/O patterns, and the basic read/write operations are *partially ordered*. Also, from our experiences in executing climate modeling applications [2, 18], we observed that some applications generate data continuously, whereas others perform this operation in single or discrete phases. This information is crucial and can be used for making clustering decisions in order to optimize the overlap between component executions.

Our scheduling model clusters the downstream components, which consume and generate data continuously as PN, and leaves the rest to behave as SDF (See Stages in Figure 1). The traversal is performed iteratively on every element of the set of temporally independent nodes. A child node becomes temporally independent once its parent finishes execution. The traversal stops if sufficient resources are not available or the heuristic decides to delay the execution of a downstream component due to data unavailability. This leaves the unscheduled downstream components waiting for resource or data availability. The clustered components are subsequently allocated and executed on distributed resources. At

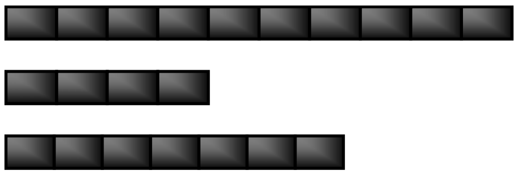


Figure 2 (a). Applications with different cost units. *Block ~ Cost unit.*

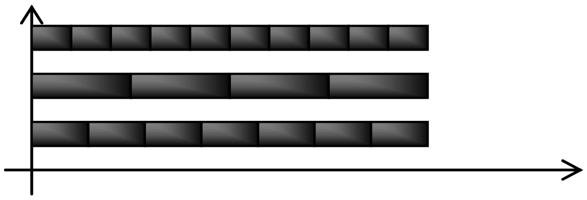


Figure 2(b). The Normalization Effect; The computation time of downstream computation is getting normalized. The total make-span is 10 cost units, however, with an additional wait time.

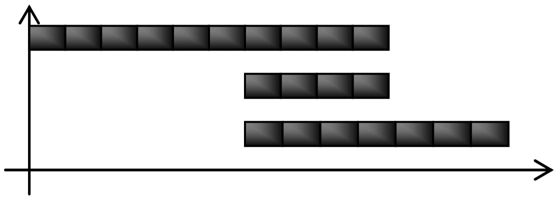


Figure 2(c). The Delay Effect. The delay in the downstream computation accrued an additional time of 3 cost units in the total make span. Nevertheless saving the CPU hours.

this point we propose that our scheduling approach is different from the conventional gang scheduling [19] as well as Gates' algorithm [14], as we assume that the components are non-pre-emptive applications and cannot be migrated once they start execution.

Ideally, if there are sufficient resources available, all clustered components can be co-scheduled; in which case, the execution time of downstream computations within the cluster would be normalized to the computation time of its parent. Figure 2(b) illustrates this scenario where jobs below are automatically stretched to the length of their respective parents. We would call this the *normalization effect* and would use it as a reference in the experiments section. This approach reduces the overall application make-span. However, it wastes CPU time because downstream computations have to wait for the data to arrive. Regardless, if no resources are available for further scheduling, then the delay is automatic. In situations where resource utilization is more important than reducing the application make-span, a delay may well be enforced explicitly. Thus, in cases where the estimated computation time of a downstream reader is lower than its parent, then it can be delayed for the difference in their execution time. This improves CPU utilization, at the expense of an increase in the application make span. We call this the *delay effect* as shown in Figure 2(c).

The third possibility is that a downstream computation is scheduled only after its parent finishes execution such as in SDF. This situation is likely to occur in cases where the upstream computation generates data in a single phase, which is generally at the end of its execution.

Depending on the possibilities described above, the application components are clustered and further allocated to distributed resources. However, in every case, the scheduling of downstream computations is delayed as much as possible by exploiting the components' inherent communication patterns. This delay is desired in order to get the latest system (network and CPU) information to generate better schedules. Grid workflows consisting of components with different write/read behaviors would specifically benefit from such an approach by optimizing the execution overlap between components, resulting in an improved workflow performance.

3.3 Component Allocation and Communication Specification

The second step involves allocation of the clustered components to best resources. This step also specifies how the scheduled components would communicate i.e. which GriddLeS IO mechanism should be implemented.

3.3.1 Component Allocation

Allocating components based on their data dependency can significantly improve application performance by reducing the communication overheads. In many scientific areas such as astronomy, geographic information systems and earth systems, data sets characterize the regions of the problem space. As a result, components processing common data sets exhibit spatial proximity. Luiz et al. [20] have demonstrated that allocating components based on such spatial proximity can reduce the application make-span. This is specifically true when the dependency between components in terms of file sharing is high. Casanova et al in [21, 22] demonstrate the efficiency of pre-staging and reusing the shared files to reduce data transfer overheads. We also base the allocation of clustered components on the similar criteria to achieve application performance as well as data and resource utilization. Once an optimum resource is chosen, then the communication behavior is dynamically specified based on current network and CPU states as described below.

3.3.2 Communication Specification

In the case of PN, there are several ways in which communication behavior can be specified. Depending on the location of the reader and writer, the IO can be redirected to a local file or remote file or a pipe. However, when remote communication is performed, a single point of network or CPU failure, which is very common in Grid environments, would require rescheduling of the components. This approach is clearly not very fault tolerant; would waste a lot of CPU time and incur additional data transfer costs. An alternative approach is to interpose a buffer at both the reader and writer's end. So, if the writing/reading is unidirectional then the writer and reader can uninterruptedly write and read data from their respective local buffers without getting affected by network failures. The proxy mechanism [23] implemented in Griddles allows such communication and also ensures data transfer and communication synchronization between the writer and reader buffers. Regardless of the time a downstream reader takes to get ready for the execution, the data which has been already generated on the writer's end can be copied using

```
Global Variables
$S_n \leftarrow |V|$ {set of components or nodes}
$R \leftarrow N$ {number of Resources}
$C_p \leftarrow C_p$ is a subset of $|E|$ {set of parallel edges}
$C_S \leftarrow C_s$ is a subset of $|E|$ {set of sequential edges}
procedure schedule
while $S_n \neq \emptyset$ d o
 $S_{sj} \leftarrow 0$      {set of sorted ready jobs}
 $S_{sj} \leftarrow$ call(getReadyJobs,NULL) {get Sorted Ready Jobs}
 $R \leftarrow R$ + call(relinquishResources,NULL)\n
 {relinquish resources from done jobs}
 $S_{BF} \leftarrow \emptyset$ {set of clustered jobs from BF traversal}
 $C_{BF} \leftarrow 0$ {total cost units for Breadth First}
 $S_{DF} \leftarrow \emptyset$ {set of clustered jobs from DF traversal}
 $C_{DF} \leftarrow 0$ {total cost units for Depth First}
 call(traverseBF, $S_{sj}$, $C_{BF}$, $S_{BF}$)

   for each $p_j \in S_{sj}$ do

   $C_{SDF} \leftarrow 0$ {total cost units for
   Depth First for single seed}
   $S_{SDF} \leftarrow \emptyset$ {set of clustered
   jobs from DF traversal for single seed}
   call(traverseDF, $p_{j,}$, $C_{SDF}$, $S_{SDF}$)
   $S_{DF} \leftarrow S_{DF} + S_{SDF}$
   $C_{DF} \leftarrow C_{DF} + C_{SDF}$
 end for

 if $C_{BF} < C_{DF}$ then
     execute $S_{DF}$
     $S_n \leftarrow S_n - S_{DF}$
 else
     execute $S_{BF}$
     $S_n \leftarrow S_n - S_{BF}$
 end if
end while
*****************************************************************

procedure traverseDF(p, ,, $C_{DF}$, $S_{DF}$)
 $J_c \leftarrow$ {q : for all $p \rightarrow q$, set of all children of p}
         for each $q \in J_c$:
     if CHILDREN(q) $\neq \emptyset$ do
         traverseDF(q,, $C_{BF}$, $S_{BF}$)
     else
         $r \leftarrow$ allocate(q)
         if x do
   $C_{DF}$ += $c$(q) {$c$: cost unit}
   $S_{DF} \leftarrow S_{DF} + q$
   $R \leftarrow R - r$ {remove chosen resource}
         end if
     end if
 end for
```

```
procedure TraverseBF ($S_{sj}$, $C_{BF}$, $S_{BF}$):
   $L_{BF} \leftarrow \{ S_{sj} \}$ {$L_{BF}$ set of BF jobs; local variable}
   while $L_{BF} \neq \emptyset$ do
      for each $p \in L_{BF}$
         $L_{BF} \leftarrow L_{BF}$ + CHILDREN(p)
         $x \leftarrow$ allocate(p)
         if x do
            $C_{BF}$ += $c$(p) {$c$: cost unit}
            $S_{BF} \leftarrow S_{BF} + p$
            $R \leftarrow R - r$ {remove chosen resource}
         end if
      end for
   end while

*****************************************************************

procedure Allocate (p)
   if !R do
      return 0
   end if
   $q \leftarrow$ PARENT(p)    {q: parent of p}
   if (Heuristic == HyBD_DELAY)&& $c(p) < c(q)$ do
      return 0
   else
      $r \leftarrow$ call(chooseResource, p, R) {select resource for p
from R}
      call(chooseCommunication, p, q)\n
      {select appropriate communication between the p and q}
      return r
   end if

*****************************************************************

Procedure getReadyJobs (NULL)
   for each p in $S_n$ do
      $q \leftarrow$ PARENT(p)
      $e \leftarrow p \rightarrow q$ {e: the edge between p and q}
      if STATUS(q) in ('done','executing') do
         if $e \in C_p$ && $R > 1$ && do
            STATUS(p) $\leftarrow$ 'ready'
         else if $e \in C_s$ && $R > 1$ && STATUS(q) = 'done' do
            STATUS(p) $\leftarrow$ 'ready'
         end if
      else if STATUS(p) == 'pending' do
         if ! PENDINGTIME(p) do
            STATUS(p) $\leftarrow$ 'ready'
      else
         STATUS(p) $\leftarrow$ STATUS(p)

      end if
   end for
   return $S_n$
```

Figure 3. Pseudo Code

specialized multi-channel file transfer mechanisms such as GridFTP [24], also supported in GriddLeS, to the reader's buffer. The reader can therefore begin execution assuming that the file is available at its end. Similarly, in the case of SDF, data generated by the writer can also be copied using such specialized copy mechanisms.

Once the components are scheduled and communication between them is specified, appropriate Grid/Web services can be invoked to perform component execution and communication. Figure 1 shows temporal instances of the workflow DAG where parts of it are scheduled as PN and the rest of them are behaving as SDF.

3.4 Scheduling Heuristics

This section describes two new scheduling heuristics, namely HyBD_MAKESPAN and HyBD_DELAY, each having a different objective function for task allocation. These heuristics are prototypes of a class of heuristics, called HyBD, which we have described earlier in this paper.

3.4.1 HyBD_MAKESPAN

HyBD_MAKESPAN focuses on minimizing the total application make-span giving less priority to CPU utilization. Accordingly, it co-schedules as many components as possible without any delay. This results in CPU overheads as some of the downstream computations hold the resource whilst waiting for the data arrival at their respective input channels, owing to the normalization effect explained earlier. However, our simulation results show that the overall make-span of the application is reduced by this heuristic at the expense of additional CPU cost.

3.4.2 HyBD_DELAY

The primary objective of HyBD_DELAY is to maximize CPU utilization at the expense of additional delay in the application make-span. Nevertheless, this approach is beneficial in situations where the CPU costs are high, in which case the component allocation can be optimized according to its computational requirements.

The pseudo code of HyBD heuristics is shown in Figure 3.

4. Implementation; GridRod

The conceptual model described above has been implemented as a workflow orchestration tool called GridRod (see Figure 4). GridRod integrates and harnesses **GriddLeS** web services and Nim**rod**/G's Grid services to perform orchestration operations. Apart from high-level operations such as component allocation and communication specification, the integration allowed leveraging of already implemented low-level orchestration services in GriddLeS and Nimrod/G. These services include job launching, job execution and communication establishment. The proposed scheduling model has been embedded in Nimrod/G and the new heuristics were added to the already existing suit of heuristics to schedule workflow components.

After the components are allocated to suitable resources, the scheduling module considers the underlying resource infrastructure (e.g. Globus [1], Condor [25]) and invokes appropriate actuation services to launch the components (Figure 4, components x, y and z) on remote resources (Figure 4, edgeJL). Likewise, the scheduling module also calls the GriddLeS web service to interpose appropriate IO mechanisms (edgeCS) for inter-component communication. For example, edgeC4 in Figure 4 represents a local file mapping, in which case, comp y and z read and write data to a local file, and the execution coherence is handled by GriddLeS. Similarly, edgeC1 represents a direct socket, edgeC2 a direct file copy and edgeC3 a proxy-based data transfer mapping. Each communication edge (edgeCn where n Є {1,2,3,4}) is specified as a logical entry in the GriddLeS Naming Service (GNS) [2]. These entries are further used by GriddLeS to interpose the specified communication mechanism. GridRod utilizes Nimrod/G as the experiment launch pad and GriddLeS as its basic communication layer to handle inter-component interaction.

5. Related Work

In this section we describe some of the related work, which complements our proposed model. However, to the best of our literature review, we could not find a direct comparison with our model, and we therefore performed an empirical analysis.

5.1 Co-scheduling Overview

The DAG co-scheduling problem is not new, and there exists a number of static [9, 26] and dynamic [25, 27] models for scheduling Grid workflow components. Grid workflow components are primarily non-preemptive applications and the data arrival and consumption times on communication links are unpredictable. This makes their scheduling on dynamic and heterogeneous platforms such as the Grid NP-Hard. Consequently, most of the scheduling effort focuses on developing the "heuristics" that targets a near optimal solution to the problem. These heuristics can be categorized as follows

- List Based – HEFT [28]
- Clustering Based – Gang Scheduling [19]
- Implicit Co-scheduling heuristics – Spin Block [29]
- Partition Based – Pegasus [30]
- First In First Out Based – GridRPC [6]
- Ordering Based – ICENI [5], Condor-DagMAN [31]
- Data dependency based – APST [21]

List based algorithms generate static schedules, which are ineffective under dynamic network and resource conditions. There is a chance, for example, that the resource on which a component was statically scheduled is no longer optimal or available at the time of component execution. Clustering-based algorithms (e.g. gang scheduling [19]) involves scheduling of pre-emptive processes, however, we assume that the workflow components are non pre-emptive applications. Similarly, implicit co-scheduling schemes, implemented primarily in cluster environments, assume that the components interact with each other by passing messages, e.g., using MPI, and by blocking and unblocking their execution, something that does not apply to our situation.

Pegasus implements a just-in-time level-based task partitioning algorithm to schedule workflow components. However, arbitrarily partitioning the graph and clustering tasks based on levels delays the execution of the whole cluster just because of one task waiting for its execution trigger. On the other hand, the FIFO-based GridRPC co-scheduling model is very simple and does not consider issues such as network and computational resource performance when making scheduling decisions.

Also, all of these scheduling models, including ICENI and APST, are designed to optimise the task allocation and have limited control over the communication specification. As mentioned earlier, such static modelling restricts the scheduling of workflow components either as SDF or PN. However, we are interested not only in dynamically allocating the components but also in specifying how and when the components should communicate, which made a direct comparison of our model with other work difficult.

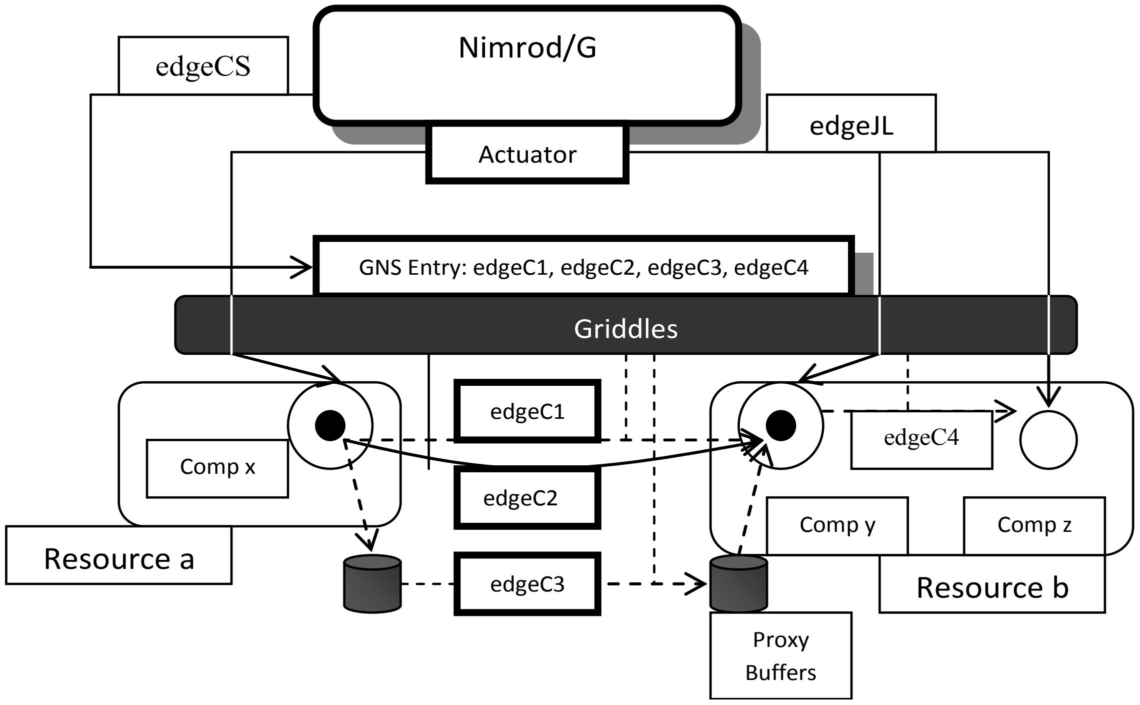


Figure 4. GridRod Architecture

An empirical experimental analysis of our scheduling model was performed. Non-iterative variants of simple Breadth First and Depth First algorithms were developed, which provided us with bounds to compare our heuristics. The non-iterative variants search the graph in one of the two orthogonal directions. However, they use the common task allocation and communication specification functions used by the HyBD heuristics.

6. Evaluation

6.1 Evaluation Metrics

The metrics against which we compared our proposed heuristics are described below:

1. Application make-span: We propose that, by exploiting the spatial and temporal parallelism between application components, the total make-span of the application can be significantly reduced. This makes application make-span a suitable metric to compare how well a heuristic explores the problem space and thereby improves the overall performance.
2. CPU utilization: We measured this metric in terms of cumulative execution time of all the components in the application. Both CPU utilization and cumulative execution time are inversely proportional to each other, which means that a high cumulative cost is equal to low CPU utilization.

6.2 Random Graph Generator

As described earlier, a workflow may have different concurrency patterns (temporal and spatial) between components, depending on their inherent communication behavior. Temporal concurrency refers to streamed pipelining whereas spatial concurrency represents parallelism. In order to generate graphs with varied concurrency patterns, we developed a random graph generator. The parameters for graph generation are the following:

6.2.1 Aspect Ratio

This parameter defines the ratio of breadth[1] vs. depth in the graph, and is specified as a value between 0 and 1. A value of 0 corresponds to a completely parallel application (breadth = number of nodes), e.g. a Parameter Sweep Application (PSA) (See figure 5(a) with breadth = number of nodes –1). On the other hand, a value of 1 corresponds to a sequential application (depth = number of nodes) (See figure 5(b)). A value of 0.5 generates the graph with a balanced depth vs. breadth ratio. The aspect ratio also determines the out-degree of the nodes, although we assume that all the nodes have unit or 1 in-degree. We are aware of the usage of the term aspect ratio in other literature such as graph drawing and image manipulation [32], and suggest that our definition is different from them. To the best of our literature search in graph generation, we could not find any terminology, which defines the aspect of breadth vs. depth, and therefore proposed our own definition.

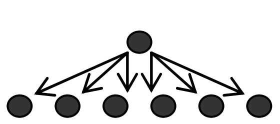


Figure 5(a)

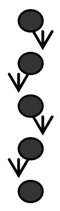
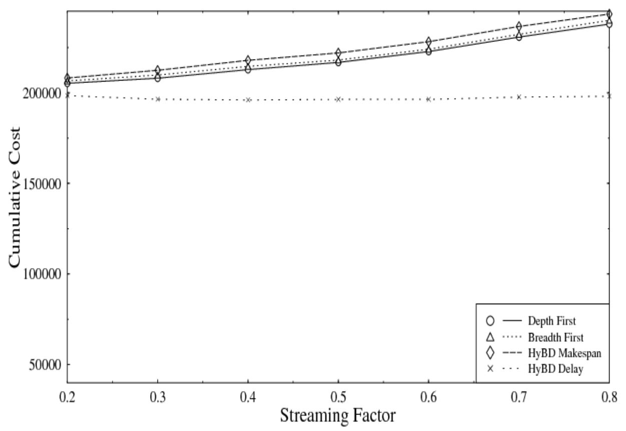


Figure 5 (b)

[1] The Breadth describes the spatial parallelism of the graph.



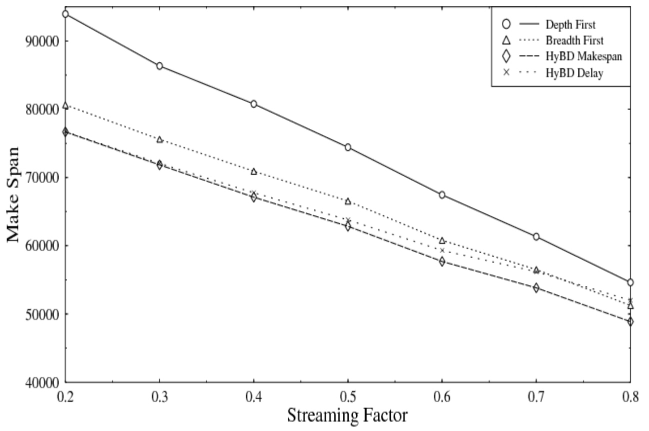
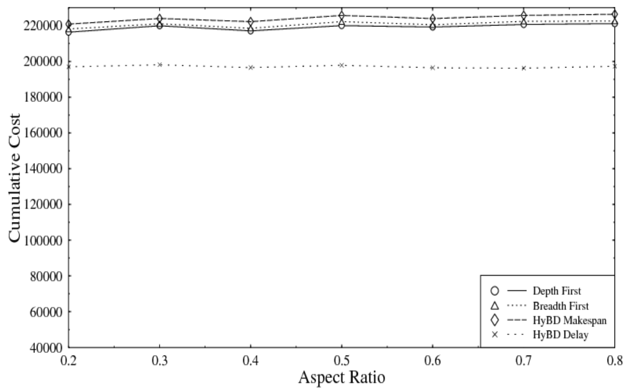


Figure 6(a). Cumulative cost vs. streaming factor **Figure 6(b). Make span vs. Streaming factor**



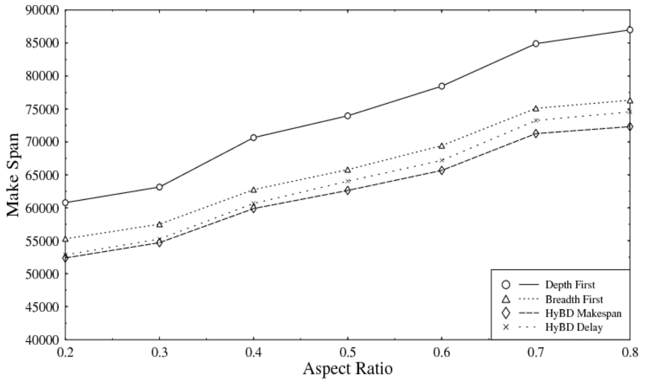
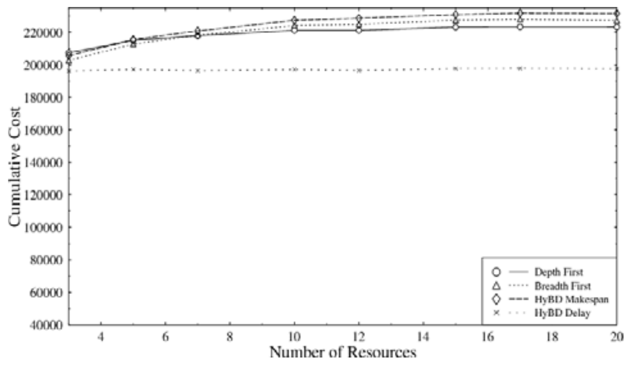


Figure 7(a). Cumulative cost vs. Aspect ratio **Figure 7(b). Make span vs. Aspect ratio**



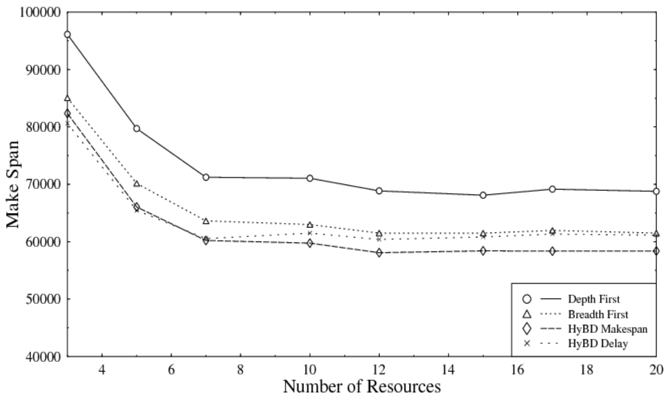


Figure 8(a). Cumulative cost vs. Number of resources **Figure 8(b). Make span vs. Number of resources**

6.2.2 Streaming Factor
This parameter is a probability measure of the streaming between workflow components. A streaming value of 0 means no streaming. All temporally non-pipelined SDF models come under this category, where a child waits for the termination of its parent to start execution. On the other hand, a value of 1 corresponds to a typical PN model where all the nodes execute concurrently. A value of 0.5 represents a symmetrically modeled workflow.

Different combinations of *aspect ratio* and *streaming factor* generate various temporal and spatial concurrency patterns in the graph, which should be appropriately exploited by the heuristics for optimizing component allocation.

6.2.3 Number of Resources
The analysis of heuristics under limited resource conditions is necessary especially when the number of tasks is relatively higher than the number of accessible resources.

6.3 The Simulator
Analyzing the behavior of scheduling models on large, heterogeneous and dynamic platforms such as a Grid is extremely difficult. The reason is that such platforms are highly unpredictable, and performing a meaningful comparison between the results obtained from real test cases is difficult. Furthermore, to perform the scalability analysis of the heuristic, a setup involving a large number of network and computational resources is required. This makes simulation an appropriate and preferable choice for conducting experimental evaluation.

In order to perform such analysis, we have developed a discrete event simulator in which our proposed heuristics are integrated. In order to simulate the Grid behavior, we used the Network Weather Services (NWS) [33] traces taken from several different resources.

7. Experiments and Results
We conducted a large set of 3,920 experiments with graphs, obtained from different combinations of parameters, namely *streaming factor*, *aspect ratio* and the *number of resources*. Such a large number of test cases prevented us from being biased towards a specific heuristic. We compared the execution time and cumulative computational costs of the tasks against each of the 3 parameters.

7.1 Execution Time
This section compares the three parameters (plotted on x-axes) against the execution time (plotted on y-axis) of the experiment. Figure 6(b) shows a declination in the execution time as the streaming between the components increases. The decline is expected as a high level of streaming provided more chances for co-scheduling, resulting in a reduced value at the y-axis. In contrast, Figure 7(b) shows a rise in the overall execution time resulting from the reduced spatial concurrency. The spatial concurrency decreased as the aspect ratio was increased. On the other hand, Figure 8 (b) shows an asymptotic behavior in the curves as the number of resources increased. The execution time was initially high owing to the low resource availability, which limited the concurrent allocation of the components. However, as the number of resources approximated the number of components, a limiting behavior in the execution time can be observed. This means that the application performance became less dependent on resource availability and more on other parameters after a threshold value.

HyBD based heuristics, when compared with breadth first and depth first always performed better. However, HyBD_DELAY always performed worse than HyBD_MAKESPAN, owing to the delays that the former incurs to optimize CPU utilization, but CPU time is saved at this expense as described next.

7.2 Cumulative Cost
Ideally, if all the components were executed independently, then their cumulative cost would remain the same on individual resources. However, owing to the normalization effect explained earlier in Section 3.2, the cumulative cost of the application actually increased. This relationship is important in order to understand the analysis described in this section.

In Figures 6 (a), 7(a) and 8(a) the cumulative computational cost for HyBD_DELAY remained the same and always lower compared to other heuristics. This is because HyBD_DELAY maximizes the resource utilization by optimizing the allocation of downstream components based on their computational requirements. Thus, a downstream component gets delayed in cases where the upstream writer has more cost units than the reader. This essentially saves time that would have been wasted whilst waiting for data arrival. However, this is at the expense of additional delays in the total make span of the application as demonstrated in Figures 6(b), 7(b) and 8(b).

8. Conclusion and Future Work
In this paper we proposed GridRod, a new service oriented workflow-scheduling tool. GridRod integrates **Grid**dLeS and Nim**rod**/G machinery for providing the orchestration services based on web-and Grid-based architectures. A new heuristic model called HyBD is proposed which exploits both spatial and temporal concurrency between workflow components for scheduling. The scheduler, in this case, not only allocates workflow components to distributed grid resources but also specifies the communication mechanism for inter-component interaction. We suggest that the runtime communication specification gives the proposed model an edge over the currently existing workflow scheduling efforts. We leverage the already existing flexible IO mechanisms provided by GriddLeS to achieve such runtime flexibility. We believe that our work opens a new field of dynamic workflow modeling and scheduling on which further research is required. We also propose two novel heuristics called HyBD_MAKESPAN and HyBD_DELAY under the HyBD model as initial steps towards the new dynamic workflow-scheduling model. As our proposed model is unlike other prevalent approaches, we performed an empirical evaluation of our heuristics. The obtained results show that both heuristics behaved consistently with their respective objective functions whilst scheduling workflow components. HyBD_MAKESPAN reduced the application make-span in all the cases whereas HyBD_DELAY always reduced the cumulative cost unit of the application components, thereby saving a significant amount of CPU time.

Our future efforts include the improvement of the current scheduling model and investigation of advanced scheduling heuristics. The objective of this investigation is to perform efficient scheduling of applications with high communication to computation ratios (CCRs). Furthermore, we are interested in integrating GridRod with advanced workflow specification tools such as Kepler for more comprehensive workflow specification, driving us to develop more descriptive models of computation for workflow orchestration.

9. REFERENCES

[1] Abramson, D., et al., *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid ?*, in *International Parallel and Distributed Processing Symposium*. 2000.

[2] Abramson, D. and J. Komineni, *A Flexible IO Scheme for Grid Workflows*, in *IPDPS-04*. 2004: New Mexico.

[3] Ilkay Altintas, A.B., Kim Baldridge,Wibke Sudholt, Mark Miller, Celine Amoreira,Yohann Potier and Bertram Ludaescher. *A Framework for the Design and Reuse of Grid Workflows*. in *Intl. Workshop on Scientific Applications on Grid Computing (SAG'04)*. 2005: Springer.

[4] *The Taverna Project*. [cited; Available from: http://taverna.sourceforge.net.

[5] *The Genie Project*. [cited; Available from: http://www.genie.ac.uk

[6] Anthony Mayer, S.M., Nathalie Furmento, Jeremy Cohen, Murtaza Gulamali, Laurie Young, Ali Afzal Contact Information, Steven Newhouse and John Darlington. *ICENI: An Integrated Grid Middleware to Support E-Science*. in *Workshop on Component Models and Systems for Grid Applications*. 2004. Saint Malo, France: Springer US.

[7] K. Seymour, H.N., S. Matsuoka, D. Dongarra, C. Lee, and H. Casanova, *GridRPC: A remote procedure call api for grid computing*, in *ICL Technical Report ICL-UT-02-06*. June 2002, Innovative Computing Laboratory, Department of Computer Science, University of Tennessee: Baltimore, MD, USA.

[8] *The VrGrads Project*. [cited; Available from: http://vgrads.rice.edu/.

[9] *The Kepler Project*. [cited; Available from: http://kepler-project.org/.

[10] Messerschmitt, E.A.L.a.D.G. *Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing*. in *IEEE Transactions on Computers*. Jan 1987.

[11] Thomas L. Adam, K.M.C., J. R. Dickson. *A comparison of list schedules for parallel processing systems*. in *Communications of the ACM*. 1974: ACM Press New York, NY, USA.

[12] Messerschmitt, E.A.L.a.D.G. *Synchronous Data Flow*. in *Proceedings of the IEEE*. 1987.

[13] Kahn., G. *The Semantics of a Simple language for Parallel Programming*. in *In Proceedings of IFIP Congress*. 1974: North Holland Publishing Company.

[14] Matias, P.B.G.a.Y. *New sampling-based summary statistics for improving approximate query answers*. in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. 1998. Seattle, Washington, United States.

[15] Charu C. Aggarwal, J.H., Jianyong Wang, Philip S. Yu. *A Framework for Clustering Evolving Data Streams* in *In Proceeings of the 29th VLDB conference*. 2003.

[16] Liang Chen Reddy, K.A., G. . *GATES: a grid-based middleware for processing distributed data streams*. in *In Proceedings of IEEE Conference on High performance Distributed Computing, 2004. Proceedings*. 4-6 June 2004: IEEE Computer Society Press.

[17] Ahmad, Y.-K.K.a.I. *Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors*. in *ACM Computing Surveys*. 1999.

[18] Anthony Mayer, S.M., Nathalie Furmento, William Lee, Steven Newhouse, John Darlington. *ICENI Dataflow and Workflow: Composition and Scheduling in Space and Time* in *Proceedings of the Workshop on Component Models and Systems for Grid Applications*. 2003. Saint Malo, France: SpringerLink.

[19] Ron Oldfield, D.K., *Applications of Parallel I/O* Oct 1996.

[20] Abramson, D., Kommineni, J., McGregor, J. and Katzfey, J. *An Atmospheric Sciences Workflow and its Implementation with Web Services*. in *The International Conference on Computational Sciences*. June 6 – 9, 2004. Krakow Poland.

[21] Jette., M.A. *Performance Characteristics of Gang Scheduling in Multiprogrammed Environments*. in *In Proceedings of the 1997 ACM/IEEE conference on Supercomputing*. Nov - 1997.

[22] Luiz Meyer, Mike Wilde, Marta Mattoso, Ian Foster. *Planning spatial workflows to optimize grid performance*. in *Distributed systems and grid computing (DSGC)*. 2006: ACM Press New York, NY, USA.

[23] Casanova, H., et al. *Heuristics for Scheduling Parameter Sweep Applications in Grid Environments*. in *In Proceedings of the 9th Heterogeneous Computing Workshop (HCW00)*. 2000.

[24] Casanova, H., et al., *The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid*, in *In Proceedings of the Super Computing Conference (SC'2000).* 2001.

[25] Abramson, J.K.a.D. *GriddLeS Enhancements and Building Virtual Applications for the GRID with Legacy Components*. in *European grid conference*. 2005. Amsterdam: Springer.

[26] Stiles, J.R., et al., *Monte Carlo simulation of neuromuscular transmitter release using MCell, a general simulator of cellular physiological processes.* Computational Neuroscience, 1998: p. 279-284.

[27] Foster, I. and C. Kesselman, *Globus: A Meta-computing Infrastructure Toolkit.* International Journal of Supercomputer Applications, 1997. **11**(2): p. 115-128.

[28] Hategan, M., et al., *GridAnt - A Client Controllable Grid Workflow System*. 2003, Argonne National Laboratory.

[29] Sarkar, V., *Partitioning and Scheduling Parallel Programs for Multiprocessors*. 1989: Paperback. 215.

[30] S. Cheng, J.S.a.K.R. *Dynamic Scheduling of Groups of Tasks with Precedence Constraints in Distributed Hard Real-Time Systems*. in *Real-Time Symposium*. December 1986.

[31] H. Topcuoglu, S.H., and M.Y. Wu. *Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing*. in *IEEE Trans. Parallel and Distributed Systems*. 2002.

[32] Andrea C. Arpaci-Dusseau, D.E.C., Alan M. Mainwaring. *Scheduling with Implicit Information in Distributed Systems*. in *Joint Conference Measurement and Modeling Computer Systems*. 1998. Madison, Wisconsin.

[33] Python xml.dom

[34] *DAGMan (Directed Acyclic Graph Manager)*.

[35] Garg, A.a.R., Adrian *Straight-Line Drawings of Binary Trees with Linear Area and Arbitrary Aspect Ratio*. in *Proceedings Graph Drawing*. 2002. Irvine, CA, USA.

[36] Rich Wolski, N.T.S., Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing in Future Generation Computer Systems. 1998.