Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 05-040

Heterogeneity-Aware Workload Distribution in Donation Based Grids

Rahul Trivedi, Abhishek Chandra, and Jon Weissman

December 28, 2005

Heterogeneity-Aware Workload Distribution in Donation Based Grids

Rahul Trivedi, Abhishek Chandra and Jon Weissman
Department of Computer Science and Engineering
University of Minnesota - Twin Cities
{trivedi, chandra, jon}@cs.umn.edu

ABSTRACT

This paper aims to explore the opportunities in porting a high-throughput Grid computing middleware to a high-performance service oriented environment. It exposes the limitations of the Grid computing middleware when operating in such a performance sensitive environment and presents ways of overcoming these limitations. We focus on exploiting the heterogeneity of the Grid resources to meet the performance requirements of services and present several approaches of work distribution to deal with this heterogeneity. We present a heuristic for finding the optimum decomposition of work and present algorithms for each of the approaches which we evaluate on a real live testbed. The results validate the heuristic and compare the performance of the different workload distribution strategies. Our results indicate that a significant improvement in performance can be achieved by making the Grid computing middleware aware of the heterogeneity in the underlying infrastructure. The results also provide some useful insights into deciding a work distribution policy depending on the status of the Grid computing environment.

1. INTRODUCTION

Grids that employ donated resources to perform its tasks have become an effective means of performing large-scale computations. One of primary projects that made use of a donation based grid was the SETI@home project [7]. Donation based grids have now found application in a diverse set of domains such as Physics [4], Weather Forecasting [5] and Medical Research [8]. These are primarily compute-oriented Grids where the amount of computation per data element is relatively high. In compute-oriented Grids the tasks can be widely dispersed irrespective of the location of the data source. Also the tasks in such a Grid computing environment execute independently with communication only between the server and the worker entities. The metric of interest in such a compute-oriented Grid is throughput, which is the total number of tasks completed in a unit of time.

Another model of Grid computation is the use of service-oriented architectures such as Grid and Web services. The union of service oriented architectures with donation based grids provides a powerful platform for performing large-scale computations, one such example being the Lattice project [6]. The critical metric for service-oriented environments is performance, which is the amount of time taken to complete an individual task. A service oriented environment has the notion of a service request which defines an explicit boundary between separate invocations of a service. Each request is composed of individual tasks all of which need to be completed within a certain time bound. The performance of the service is a measure of its response time for an individual service request.

The challenge in hosting such a service on heterogeneous set of resources is maximizing the performance of the service by intelligent scheduling of tasks on the Grid. Our analysis is based on BOINC [2] which is a widely used Grid computing middleware. BOINC has a centralized server which hands out tasks to the workers. The BOINC server scheduler ignores the heterogeneity of the workers when distributing the tasks. This affects the service performance as the response time of a service request is the time taken to complete all the individual tasks in a request. Hence the response time of the service request is bottlenecked by the slowest node in the Grid.

In this paper we explore several workload distribution strategies that make use of the heterogeneity information of the Grid resources to make better scheduling decisions. The aim of these workload distribution strategies is to distribute the workload proportional to the capabilities of the nodes in the Grid. We focus on the computation and communication capabilities of the worker nodes. We first propose a workload distribution strategy that does proportional allocation of work by decomposing each task into finer sub-tasks so that the faster nodes in the Grid perform more work. We then propose strategies that make use of historical information to estimate the capability of the worker nodes and then use this information to create and assign tasks that match the capability of the Grid nodes. We evaluate the workload distribution strategies on PlanetLab [1], a planetary scale distributed testbed. We have used BLAST [3], an exemplar service in the domain of bioinformatics, as a test case since it represents emerging large-scale data rich services.

The rest of this paper is organized as follows: In section 2 we expose the pertinence of this problem which also forms the motivation of our work. In section 3 we propose different strategies to exploit the heterogeneity of the resources. Section 4 gives a performance evaluation and comparison of the different strategies and provides insights into the applicability of each of the strategies. Finally in section 5 we summarize our results and list our future research directions.

2. MOTIVATION

A non-dedicated and distributed Grid such as in a typical deployment of BOINC is characterized by its use of multiple nodes with varying computational capabilities. Different worker nodes in such an infrastructure typically have different CPU speeds, memory and disk capacities. Moreover the nodes have different connection speeds and their bandwidth to the server node is also dependent on their geographical location. In this section, we exhibit the heterogeneity among the worker nodes using our test application as the benchmark.

2.1 System Model

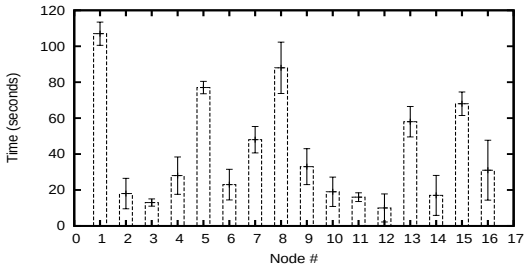
The BOINC system consists of single centralized scheduler which consists of two major components. A scheduling server which hands out the tasks to the worker nodes and a data server which manages the transfer of the input and output files from the server to the worker nodes. The scheduling server and the data server are co-located on the same server node. The BOINC system is a pull-model where the worker nodes poll the BOINC server periodically, requesting work. The worker nodes after completing the computation send back the output files to the server. Each of the tasks that are handed out by the server are individual and require no interaction between the worker nodes. Hence the only communication is between the server and the worker nodes.

The application is a modification of the standalone bioinformatics application called BLAST (Basic Local Alignment Search Tool) which runs as BOINC service. BLAST is an algorithm for rapid searching of DNA and protein databases. The BLAST algorithm compares novel DNA sequences with previously characterized genes, and is used to identify the function of the newly discovered proteins. BLAST takes an input sequence and compares it to a formatted database file and generates an output file containing a similarity score and similarity matches with the database. The BLAST application serves as a good representative of Grid service as it is both computationally heavy and data-rich, as it requires the transfer of a large amount of data to perform the computation.

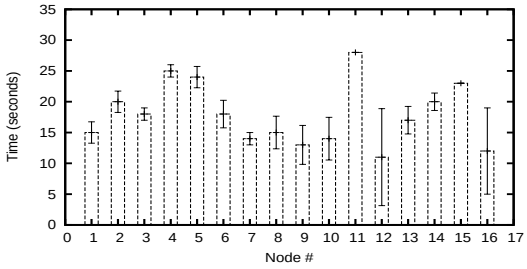
The worker nodes were a set of 16 randomly selected nodes from the PlanetLab infrastructure. In our BOINC setup we use a 119 MB formatted file of sequences (drosoph.nt) as the BLAST database of gene sequences. The input sequence used for comparison was a randomly-selected sequence from the database; the sequence was of length 569 bytes. Each of the 16 worker nodes was given an equal share of the work by splitting the database into 16 equal-sized chunks. On each run the database chunk was transferred from the BOINC server to the worker node. The worker nodes after completing the BLAST computation return the result back to the BOINC server. The communication time is largely dominated by the transfer of the database chunk as the input sequence file and the resultant output file are comparatively much smaller in size.

2.2 Heterogeneity Results

Figures 1(a) and (b) plot the average per-node computation and communication time over multiple runs along with the standard deviation. Figure 1(a) clearly shows the wide diversity in the computational capability of different nodes with the slowest node being almost 10 times slower than the fastest node in the grid. For instance while node 12 only takes about 10 seconds on average for its computation, node 1 takes about 107 seconds to do the same amount of computation. However, Figure 1(b) shows that the average communication times are highly correlated across nodes with the fastest node only being about twice as fast as the slowest node.

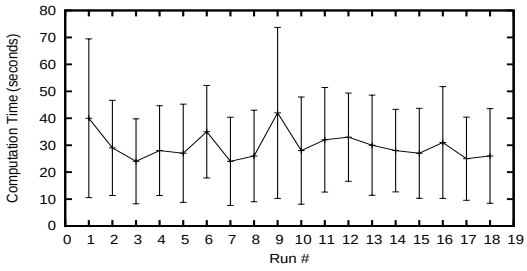


(a) Computation Time

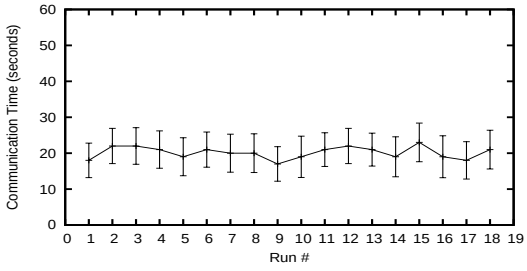


(b) Communication Time

Figure 1: Per-node average computation and communication time. The error bars represent standard deviation



(a) Computation Time



(b) Communication Time

Figure 2: Inter-node variability in computation and communication time over multiple runs

Figures 2(a) and (b) depict the the average computation and communication time for multiple runs. The errors bars in the graph represent standard deviation of the time across nodes for each run.

Another interesting observation we make from Figures 1(a) and 2(a) is the difference between the inter-node vs. the intra-node variation in computation time. The large values of the standard deviation from Figure 2(a) indicate a large inter-node variation in computation time even over the same run while the tight standard deviation in Figure 1(a) imply small intra-node variation even across multiple runs. This is a useful observation as it suggests that it might be easier to distinguish between different node capabilities, thus exploiting Grid heterogeneity without having to worry about dynamic intra-node variations. While from figures 1(b) and 2(b) we observe that the inter-node and the intra-node communication difference is not very large. Hence it is not required to handle the communication heterogeneity as much as the computation heterogeneity.

From the observations made in this section it is clear that the baseline BOINC workload distribution, which ignores heterogeneity of the worker nodes, is not suitable for a service-oriented setup as the heterogeneity in the underlying infrastructure could be exploited to improve the service performance. Hence a more intelligent scheme of workload distribution is necessary which takes into account the heterogeneity of the nodes within the infrastructure. In the future sections we explore various workload distribution strategies and evaluate their performance under different scenarios.

3. WORKLOAD DISTRIBUTION STRATEGIES

In the above section 2, we observed that the nodes in a typical donation based Grid exhibit substantial heterogeneity. The default BOINC scheduling policy ignores the heterogeneity of the nodes when handing out tasks to the worker nodes. In our BOINC Grid setup the tasks are subdivided at the server and once all the results are obtained they are then merged to generate the final output. Due to the heterogeneity amongst the worker nodes the slowest node in the Grid becomes the bottleneck and affects the total response time of the task. Thus more intelligent workload distribution strategies would increase the performance of the service.

In this section, we describe workload distribution strategies that could be applied to the BOINC Grid infrastructure. The workload distribution strategies fall into two main categories as follows:

- Equal-Size Workunit Allocation
- Variable-Size Workunit Allocation

3.1 Equal-Size Workunit Allocation

The Equal-Size Workunit Allocation strategy creates workunits of equal sizes and lets the worker nodes pick up any of the workunits that need to be executed. Thus it does not make any distinction between the worker nodes. The two approaches to this workunit allocation strategy are as follows:

- Coarse-Grained Workunit Allocation
- Fine-Grained Workunit Allocation

3.1.1 Coarse-Grained Workunit Allocation

The Coarse-Grained Workunit Allocation strategy creates as many workunits as the number of nodes in the grid. This is the baseline BOINC workload distribution and is the simplest scheme of workunit allocation. We have already seen how this workunit allocation strategy suffers from bad response times due to the heterogeneity amongst the nodes in the Grid.

3.1.2 Fine-Grained Workunit Allocation

The Fine-Grained Workunit Allocation strategy aims to tackle the heterogeneity amongst the worker nodes by subdividing the workunits into finer chunks. This strategy leads to a better workload distribution because with finer workunits the faster nodes in the Grid pick up more workunits from the server than the slower nodes in the Grid. This load-balances the system thereby reducing the overall response time.

Heuristic for Workload Decomposition

In this section we present a heuristic which estimates the optimum decomposition for a given problem. The main idea behind the Fine-Grained workload allocation is to load balance the Grid by creating finer units of work. The Grid gets better load balanced as the work is decomposed into finer units, decreasing the granularity of the units of work at each step. But this decomposition into finer units of work will exhibit an improvement in performance only while the Grid is not perfectly load balanced. Once the Grid is perfectly load balanced a further decomposition of work will not lead to an improvement in performance. Also in some cases a further decomposition of work may actually bring in overheads that might negatively affect the performance.

We now present an algorithm which finds the optimum decomposition of work for a given problem-size:

Algorithm 1 Workload-Decomposition(*Problem*$_{size}$, *Init*$_{decomp}$ /* Initial Decomposition */, *step* /* Factor by which the granularity is to be decreased */, δ /* Limiting condition for the granularity */)

1: Current$_{decomp} \leftarrow$ Init$_{decomp}$
2: Old$_{granularity} \leftarrow \frac{Problem_{size}}{Init_{decomp}}$
3: **while** TRUE **do**
4: Current$_{decomp} \leftarrow$ Current$_{decomp}$ + step
5: New$_{granularity} \leftarrow \frac{Problem_{size}}{Current_{decomp}}$
6: **if** Old$_{granularity}$ - New$_{granularity} <= \delta$ **then**
7: Optimum$_{size} \leftarrow$ Old$_{granularity}$
8: Return Optimum$_{size}$
9: **else**
10: Old$_{granularity} \leftarrow$ New$_{granularity}$
11: **end if**
12: **end while**
13: End

The above algorithm states that as the work is decomposed into finer units of work an improvement in the performance will be observed only while the difference between the granularity of work from one step to the next is above some constant δ. The value of this constant δ is specific to the problem or the application. The point where the difference in the granularity drops below this constant δ is the optimum decomposition for the current problem.

We now present an application of the above algorithm for finding the optimum decomposition of work for the BLAST application given a Database size and a Grid size.

In case of the BLAST application -
Problem$_{size}$ = Database$_{size}$
Init$_{decomp}$ = Grid$_{size}$
The decomposition is the number of workunits *Num*$_{wus}$ which is initially set to the Grid size. The granularity of work is the size of the fine-grained database chunk.
Granularity = $\frac{Database_{size}}{Num_{wus}}$
The granularity of the workunits is decreased in each iteration by increasing the number of workunits in orders of *step*.

In case of the BLAST application the dominant component costs in the total time are the communication and computation costs which are dependent on the size of the database chunk. The heuristic states that an improvement in the total time will be observed as the number of workunits are increased in orders of *step* only until the difference in the size of the database chunk between the two configurations is greater than δ_{step}, which is fixed for a certain *step* value. When the difference in the size of the database chunk reduces to less than δ_{step} a further improvement in the total time will not be observed. The number of workunits here Num$_{wus}$ are the optimum number of workunits with the size of the database chunk being the optimum size. The starting value for Num$_{wus}$ is the Grid size (Num$_{wus}$ = Grid size is the coarse-grained workunit allocation strategy).

The relation between step and δ_{step} is as follows:

step	δ_{step}
4	0.4 MB
8	0.8 MB
16	1.6 MB

Minimum step value(order of increasing workunits) is chosen as 4 due to the nature of the BLAST databases. The database file consists of a list of gene sequences. When splitting the database it is not possible to arbitrarily split the database in between a sequence and hence the split has to be aligned to the start or end of the sequence. This introduces some deviation from the expected database chunk size. A minimum granularity of 4 is chosen to accommodate for that error. The choice of this step value would depend on the Grid environment. In most cases we would like to choose the lowest possible step value (which is 4) to get the best granularity of work. Though in some scenarios, such as when the database size is very large, a larger step value could be chosen to reduce the search space.

3.2 Variable-Size Workunit Allocation

The Fine-Grained Equal Size workunit allocation strategy requires the worker nodes to return to the server every time to fetch additional workunits. The Variable Size workunit allocation strategies attempt to eliminate this overhead by creating and handing out variable sized workunits by matching the size of a workunit assigned to the relative capability of the node in the Grid. The server does selective scheduling by forcing a worker node to pick up a specific workunit. This workunit allocation strategy requires a method of finding the capabilities of the nodes in the Grid in order to create these different sized workunits. We have employed two different approaches to deciding the size of the workunits to be assigned to the nodes in the Grid. The two approaches are as follows:

- Using the benchmark information collected by BOINC
- Using the historical information of workunit distribution observed in the Fine-Grained workunit allocation case
- Using the observed computation and communication durations of each node in the Grid

3.2.1 *Variable Size Allocation using BOINC Benchmark Information*

The BOINC core client collects benchmark information when the core client is executed for the first time on the worker node. This information is updated at periodic intervals and is reported back to the BOINC server on every work request. The BOINC server maintains this information in the server database for each of the worker nodes. In this section we try to estimate the compute capability of a node as a function of two of the benchmark parameters, viz. Fpops (Floating point operations per second) and CPU-efficiency. The Fpops value is calculated using the Whetstone benchmarks. CPU-efficiency estimates the amount of CPU time a BOINC application gets for each wall-clock second that it is run. This indirectly is a measure of the load on a worker node. The communication capability information of a worker node does not need to be estimated as it is available on the BOINC server as a measure of its download/upload bandwidth. In figure 3 we plot the observed compute capability of 16 Grid nodes to their Fpops and CPU-efficiency values. We observe from the figure that the data points are widely scatterred. Standard interpolation techniques for surface plotting when applied to the observed data fail to provide an estimate, within reasonable error, of the compute capability of a new node as a function of these two parameters.

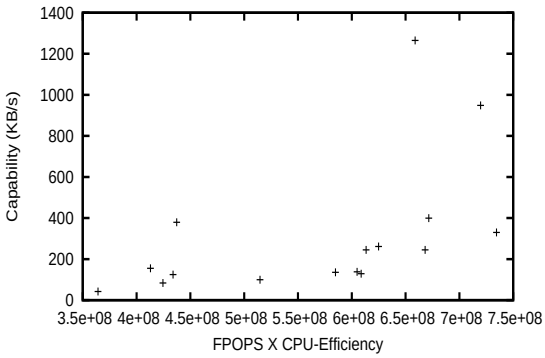


Figure 3: Compute Capability as a function of Fpops and CPU-efficiency

The BOINC collected benchmark information is thus insufficient to estimate the capability of a Grid node. In the next two sections we explore two approaches to variable size allocation that make use of information from the past to estimate the node capabilities which is then used to decide the size of the database chunks to be assigned to each of the Grid nodes.

3.2.2 *Variable Size Allocation using Observed Node Capabilities*

This Variable Size workunit allocation strategy makes use of the observed node capabilities viz. computation and communication durations to decide the size of the workunits to be allocated to nodes in the Grid. This strategy collects the computation and communication duration information for each node from the coarse-grained equal size workunit allocation strategy to get an estimate of the node capability. The capabilities of all the nodes within the Grid

are used to decide the proportional share of the database to be assigned to each node which form the variable size workunits. The algorithm for finding the variable size workunit allocation is as follows:

Algorithm 2 Var-size-Observed (DB_{size}, $Grid_{size}$, *Nodes[]* List of Nodes in the Grid, *CNC[]* Compute + Communication Times Of Grid Nodes)

1: FixedChunk$_{size} \leftarrow \frac{DB_{size}}{Grid_{size}}$
2: **for all** node in Nodes[] **do**
3: Capability[node] $\leftarrow \frac{FixedChunk_{size}}{CNC[node]}$
4: **end for**
5: Total$_{Capability} \leftarrow \sum_{n=1}^{Grid_{size}} Capability[n]$
6: Proportion$_{factor} \leftarrow \frac{DB_{size}}{Total_{Capability}}$
7: **for all** node in Nodes[] **do**
8: VariableChunk$_{size}$[node] $\leftarrow$ Capability[node]$\cdot$Proportion$_{factor}$
9: **end for**
10: Return

3.2.3 *Variable Size Allocation using Historical Workunit Distribution Information*

The Fine-Grained workunit allocation strategy does load balancing by subdividing the workunits into finer chunks. The heuristic for fine-grained workunit allocation gives a method of finding the optimum number of workunits for a particular database size and grid size. This variable size allocation strategy makes use of historical information of the workunit distribution observed in the Fine-Grained workunit allocation case to deciding the size of the variable size workunits. The algorithm for finding the variable size workunit allocation is as follows:

Algorithm 3 Var-size-Historical (DB_{size}, $Grid_{size}$, *Nodes[]* List of Nodes in the Grid)

1: Optimum$_{workunits}$$\leftarrow$Fine-Grained-Heuristic(DB$_{size}$, Grid$_{size}$) /* Apply the heuristic function for Fine-Grained Allocation Strategy to obtain the optimum number of workunits for this configuration */
2: Obtain Response$_{times}$[] with Fine Grained Workunit Distribution for Optimum$_{workunits}$
3: WU$_{distr}$[Grid$_{size}$]$\leftarrow$Workunit-Distribution(MIN (Response$_{times}$[])) /* Get the workunit distribution for the best response time*/
/*Use the workunit distribution to obtain the variable size chunk for each node*/
4: **for all** node in Nodes[] **do**
5: VariableChunk$_{size}$[node]$\leftarrow$Group-Database-Chunks(WU$_{distr}$[node])
6: **end for**
7: Return

4. EVALUATION

In this section, we validate the heuristic presented in section 3.1.2 and evaluate the performance of the workload distribution strategies discussed in the previous section. We first describe our BOINC Grid setup and experimental details followed by the performance results.

4.1 Experimental Methodology

BOINC Grid Setup

We run our Grid on Planetlab - a shared distributed infrastructure consisting of donated machines. Our experimental setup consists of 16 Planetlab nodes. The Planetlab nodes serve as the Grid worker nodes. The BOINC Grid server runs on a dedicated machine that is outside the Planetlab infrastructure. Each Planetlab node runs the Fedora Core 2 Linux kernel 2.6.8 and has 5GB of disk space. The nodes have varying hardware capabilities and are geographically distributed. Most of the worker node CPU's are Pentium III or Pentium 4 with CPU speeds in the range from 1.2 GHz to 3.0 Ghz. The amount of memory on each of the nodes is between 1GB and 2GB. We used the BOINC development version 4.72 to setup our Grid prototype on the Planetlab testbed.

We used the BLAST (Basic Local Alignment Search Tool) bioinformatics application as described in section 2 to run as a service on top of our Grid prototype. In our setup, BLAST is modified to run as a BOINC project: it is hosted on the BOINC server which hands out the application workunits to the worker nodes for computation. In this setup, the BLAST executable is kept unmodified and a BOINC-specific wrapper is written around it. A workunit consists of an input sequence and a portion of the BLAST database provided as input files. The result of each workunit execution is an output file containing a similarity score generated by the BLAST code. The BLAST computation at each worker node is performed in two steps. The first step consists of formatting the database using a BLAST command 'formatdb', after which the actual sequence comparison is performed to yield a result file. In our BOINC setup, the results are sent back to the server which merges them together into a single output file. We use two formatted databases, one of size 119MB (drosoph.nt) and other of size 284MB (sts). The input sequence used for comparison was a randomly-selected sequence from the database; the input sequence length was 569 bytes. The BOINC workunits for the BLAST service are generated by splitting the database into chunks.

We conducted our experiments by executing multiple BLAST requests on our testbed and measuring the total request execution times along with the component costs such as computation and communication times at each worker node. Each requests consists of the BLAST execution for a single input sequence and the whole BLAST database. The Planetlab infrastructure being a very dynamic environment we conducted our tests in a cyclic manner in order to smoothen out the effects of temporary variations in computation and communication load. The different configurations in the Fine-Grained equal size workload allocation were tested in an interleaved manner. A similar test setup was used for the comparison of the Equal sized and Variable size allocation schemes. Also the tests were repeated at different times during the day. We now present the experimental results and their implications in choosing a workload distribution strategy.

4.2 Comparison of Equal-Size Workunit Allocation Strategies

In this section, we compare the two equal-sized workunit allocation strategies viz. Coarse-Grained vs. Fine-Grained. The Coarse-Grained workunit allocation strategy is the default execution model of BOINC.

From Figure 4 it is clear that a significant improvement in the total response time is possible just by the simple scheme of creating finer-grained workunits to do better workload distribution. One other point to note from Figure 4 is that the performance improve-

ment from the coarse-grained to the fine-grained workunit allocation strategy is greater when the total size of the database is larger. The reason behind this behavior is that for a larger database size, for the coarse-grained workunit allocation there is greater disparity among the worker nodes. Hence with the fine-grained workunit allocation a greater improvement in the total time is observed as there is much more scope of load-balancing the system for a larger database.

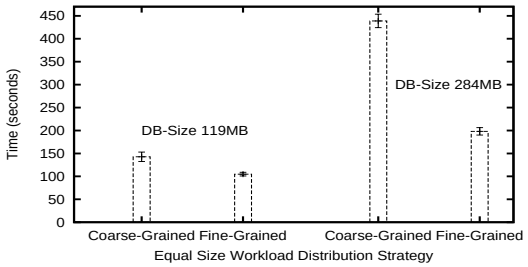


Figure 4: Comparison of Equal-Size Workunit Allocation Strategies for databases of size 119MB and 284MB

4.3 Fine-Grained Workunit Allocation Strategy

The Fine-Grained workunit allocation strategy achieves better workload distribution by creating finer-grained workunits because of which the faster nodes in the Grid end up executing more number of workunits. The Figure 5 illustrates how accurately the capabilities of a node match against the number of workunits executed by it. The capability of a node is expressed in units of KB/s. This node capability is the compute + communication capability of the node.

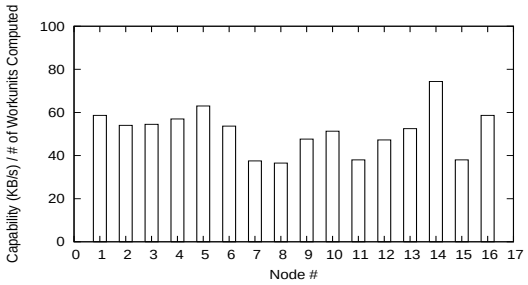


Figure 5: Ratio of node capability to workunit distribution for DB of size 119 MB

From Figure 5 we observe that the ratio of node capability to the number of workunits executed by that node is almost the same. This shows that the fine-grained workunit allocation better load-balances the Grid nodes. Note here that the size of each workunit differs for the two different databases. These results are further impetus to the variable size workload distribution described in section 3.2.2 which uses the same capability values to decide the size of the workunits to be assigned to each node.

4.3.1 Heuristic Validation

In section 3.1.2 we presented a heuristic which for a given database size predicts the optimum number of workunits that would give the lowest possible response time. In this section, we validate this heuristic under different configurations. We also present here a breakdown of the total time to show the component costs such as the computation, communication and overhead times. The total time is represented as follows-

$$Total_t = MAX(\forall G_n(P_t + Oh_t + \sum_{Workunits} (Cm_t + Cp_t)))$$

where,
Total$_t$ - Total Time
G$_n$ - Grid Nodes
Cm$_t$ - Communication Time,
Cp$_t$ - Computation Time,
P$_t$ - The preamble time is the time taken to create the workunits at the server
Oh$_t$ - Overhead Time - the amount of time that the worker node sits idle while the result of one workunit is uploaded to the server and the download of the next workunit begins.

The total time for one node is thus the sum of the communication and computation times for all the workunits executed by that node plus the overhead and the preamble time. The total time taken for that request is thus the time taken by the slowest node, which is the maximum of all the per node times. The overhead time is bound to rise as the number of workunits are increased. In the figures below the miscellaneous time is the preamble time for creating the workunits at the server and the amount of time the first set of workunits wait at the server before being picked up by a worker node.

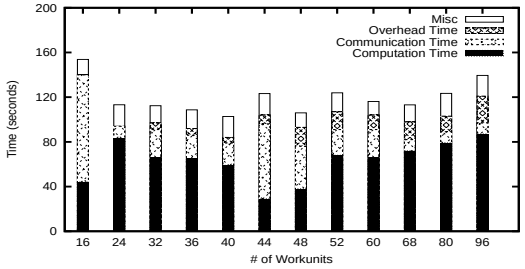


Figure 6: Effect of varying number of workunits for database of size 119 MB

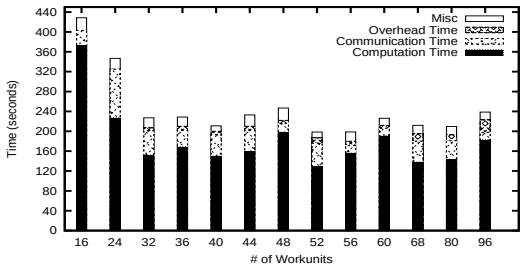


Figure 7: Effect of varying number of workunits for database of size 284 MB

Figures 6 and 7 show the performance of the service for varying number of workunits for databases of size 119 MB and 284 MB. We also show the component costs in each of the configurations. These breakdown times are of the slowest node in the Grid i.e. the

bottleneck node in the Grid. Overall from the component costs we see that the dominant cost is the computation time. As the number of workunits are increased we get a better load distribution among the grid nodes and hence the computation time reduces.

For the database of size 119 MB the optimum number of workunits is **40** as obtained from the heuristic. The best response time as depicted in Figure 6 is seen for 40 workunits. We see that the beyond these number of workunits the total response time flattens out and then rises gently. Hence the heuristic states the point where the curve starts flattening out thus indicating that beyond that point no further improvement in the total time is possible even if the number of workunits are increased. This is the optimal configuration because for these number of workunits the Grid is load-balanced and increasing the workunits further will not offer any further improvement. The gentle rise in the total time for very large number of workunits such as 80 and 96 is because for these number of workunits the overhead time begins to affect the total time.

Figure 7 shows the effect of varying number of workunit for database of size 284 MB. For this database size the optimum number of workunits is **56**. A similar graph is observed for this database size with the lowest response time seen for 56 workunits. The computation time is seen to be lowest for these number of workunits. Also the overhead costs are still small and hence do not affect the total time.

4.3.2 Effect of Increasing the Grid Size

In this section, we study the effect of increasing the Grid size from 16 worker nodes to 32 on the heuristic. Figures 8 and 9 show the effect of varying the number of workunits for database of size 119 MB and 284 MB for a Grid of size 32.

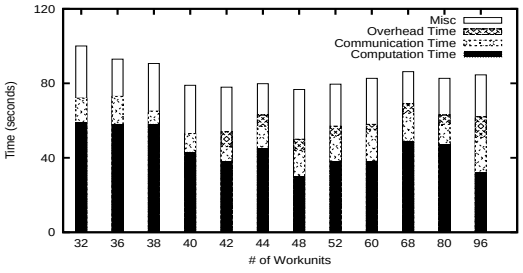


Figure 8: Effect of varying number of workunits for database of size 119 MB and Grid size 32

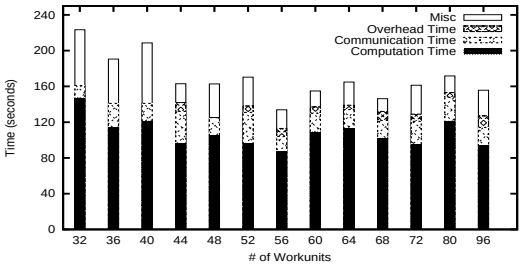


Figure 9: Effect of varying number of workunits for database of size 284 MB and Grid size 32

For the database of size 119 MB the optimum number of workunits is 40 and for the database of size 284 MB the optimum number of workunits is 56. The first point to observe when the Grid size is increase from 16 nodes to 32 nodes is that the performance gain from the coarse-grained to the fine-grained workunit allocation is lesser. The reason as explained in section 4.2 is that with a larger Grid size the size of each database chunk for the coarse-grained workunit allocation is small. Hence the heterogeneity of the nodes does not get exposed as much which leaves lesser room for load-balancing. Comparing the Figures 7 and 9 we see that for the database of size 284 MB the performance gain with 16 Grid nodes was about 230 seconds while that with 32 Grid nodes is about 77 seconds.

Also with a 32 node Grid it is observed that beyond the optimum number of workunits the total time remains nearly the same for many more 4 workunit step increments. This is because when the Grid size is 32, workunit increments of 4 does not greatly change the load-balancing within the Grid nodes and hence the total time observed remains the same as the number of workunits are increased. The miscellaneous time is seen to increase for a larger Grid size. This component cost is the preamble time for creating the workunits at the server and the amount of time a workunit waits at the server before being picked up by a worker node. With a larger Grid size there are more number of worker node requests coming into the server which increases this component cost. Variation in this component cost is observed because this cost depends on which workunit is picked up by the bottleneck node in the Grid. As the workunits are created sequentially if the bottleneck node picks up one of the earlier created workunits then the miscellaneous component cost is small and for the later workunits is large. As the decision of which workunit is picked up by a worker node is decided by the BOINC scheduler and is out of our control this component cost cannot be perfectly characterized.

4.3.3 Effect of Increasing the Input Sequence Size

In this section, we study the effect of increasing the input sequence size on the performance of the service. We also validate the fine-grained workunit allocation heuristic for this larger input sequence. Uptilt now for all our experiments we used an input sequence of size 569 bytes. The input sequence is a randomly-selected sequence from the database. We now present the effects of increasing the input sequence size to 27 KB. The input sequence is still a randomly-generated sequence from the database.

Figures 10 and 11 show the effect of increasing the input sequence size on databases of size 119 MB and 284 MB. From the heuristic described in section 3.1.2 the optimum number of workunits for the 119 MB database is 40 and the optimum number of workunits for the 248 MB database is 56. For both the database sizes we observe that the heuristic does apply.

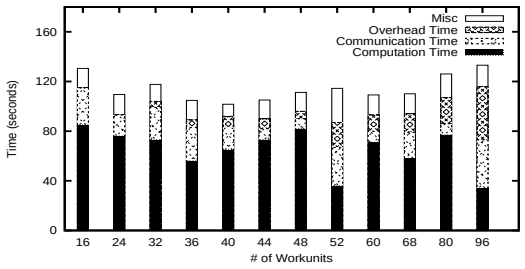


Figure 10: Effect of varying number of workunits for database of size 119 MB and Input Sequence of Size 27KB

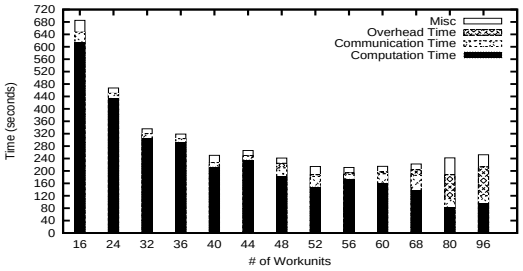


Figure 11: Effect of varying number of workunits for database of size 284 MB and Input Sequence of Size 27KB

In case of the 119 MB database from Figure 10 we observe that the nature of the graph is similar to that with input sequence size of 569 bytes. The graph slopes gently from 16 to 40 workunits beyond which it remains flat upto 80 workunits. At this point the overhead time begins to affect the total time which then starts rising.

In case of 284 MB database we observe that with 16 workunits (Coarse-Grained workunit allocation) the total time taken is larger with the 27 KB input sequence than the 569 byte input sequence by about 250 seconds with the dominant cost being the computation time. As the number of workunits are increased we observe that with better load balancing the computation cost reduces and the optimum total time is observed at 56 workunits. Thus in this case though the total time is observed to be very high with 16 workunits it falls sharply and the minimum total time observed at 56 workunits which matches the heuristic value.

4.4 Comparison of Equal-Size and Variable-Size Allocation Strategies

In this section, we compare the Equal-Size and Variable-Size allocation strategies we described in section 3. In the previous section we evaluated the Fine-Grained Equal Size workunit allocation strategy and validate that heuristic. As we have seen from the component costs there is an overhead associated with a worker node going back to the server every time to fetch additional workunits. Hence the intuition behind the variable-size workunit allocation strategies is to get rid of the overhead costs by grouping the workunits initially at the server and handing them out to each worker node. This would load-balance the system and would also take away the overhead of returning to the server every time for fetching additional work.

In Table 1 we compare the workload distribution strategies for different database sizes and Grid sizes. The *Historical* and the *Observed* workload distribution strategies are the variable-size workunit allocation strategies described in sections 3.2.3 and 3.2.2.

We observe that the results obtained are different from our intuition as for 3 out of the 4 cases the fine-grained equal size allocation strategy does better than the variable-size allocation strategies. The only case in which the variable-size allocation strategy performs equal or slightly better than the fine-grained equal size allocation strategy is for the database of size 119 MB and Grid size of 16.

The explanation for the observed results is as follows:
In our BOINC Grid setup we have a single server which is handing out workunits to the worker nodes. The entire database is maintained at this server with the scheduling server and the data server both running on the same sever node. The worker nodes continuously query the BOINC server requesting work. The input files for a computation are the input sequence and the database chunk. The database chunk size is of the order of MB's and hence it dominates the communication cost.

		Equal-Size (secs)		Variable-Size (secs)	
DB-size	Grid-size	Coarse	Fine	Historical	Observed
119MB	16	153	102	102	96
284MB	16	428	198	227	244
119MB	32	100	78	84	90
284MB	32	223	133	168	146

Table 1: Comparison of Equal Size and Variable Size Workload Allocation Strategies

The worker nodes are constantly polling the BOINC server for work as soon as workunits are created at the server the BOINC scheduler hands out these workunits to the worker nodes. Thus all the worker nodes start downloading the input files at the same instant of time. This clogs the download bandwidth at the server and hence all the worker nodes take longer to download the input files than the standard download time that would be observed if there was just one worker node downloading the input files at one time. *This contention at the BOINC server affects the total time*. The variable-size workunit allocation strategies take into account the heterogeneity of the worker nodes and create different sized workunits so as to take away the overheads of the fine-grained allocation strategy. But due to the contention at the server the communication time increases which increases the total time. The Fine-Grained workunit allocation strategy gives better results because with this strategy the worker nodes download the input files from the server at the same time only for the first workunit. For all the future workunits the worker nodes return to the server at different times due to which there is lesser contention for the future workunits. Thus the worker requests to the server get *temporally spaced* which reduces the contention and thus reduces the total communication time. Thus in the Fine-Grained allocation strategy the heterogeneity of the nodes helps reduce the total response time.

This also provides an explanation for the heuristic presented in section 3.1.2. At the optimum number of workunits the system is perfectly load-balanced. Beyond this point with more number of workunits the heterogeneity of the nodes does not exposed as much and hence the total response time does not drop. The contention affects the communication time and the overhead costs also begin to dominate.

From Table 1 we see that the only case in which the variable-size does equal or better than the equal-size workunit allocation is for the database of size 119 MB and Grid size 16. This is because for this database size and Grid size the effect of the contention at the server is least among the four cases and hence the variable-size allocation strategies do better. For all other cases, increasing the DB size to 284 MB or increasing the Grid size to 32 adds to the contention at the server due to which the total time increases and hence the fine-grained allocation strategy does better.

4.4.1 Effect of Data Contention at the Server

In the previous section, we observed that the variable-size workunit allocation strategies do not perform a whole lot better than the fine-

grained equal-size allocation strategies due to the data contention at the server.

Figure 12 shows the complete distribution of 40 workunits among the 16 worker nodes for DB of total size 119 MB. From the figure we see that the first workunit takes more amount of time to download and the future workunits take lesser time. This is because contention at the server is more for the first workunit as all the worker nodes are download the input files at the same time.

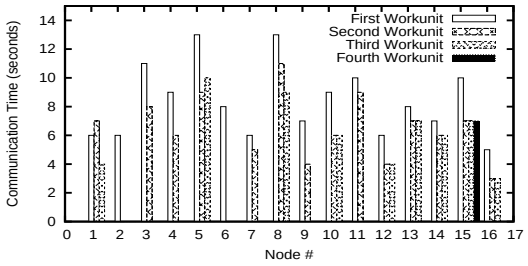


Figure 12: Effect of contention at the server for DB size 119 MB and Grid size 16

Figure 13 shows the comparison of the total communication time for the equal-size fine-grained allocation case to the communication time in the variable-size allocation case. From the figure we see that for most of the nodes the two values are the same. We observed in the results for database of size 119 MB and Grid of size 16 in Table 1 that the variable-size historical and the fine-grained allocation case have nearly equal total times which is reflective of the results seen here.

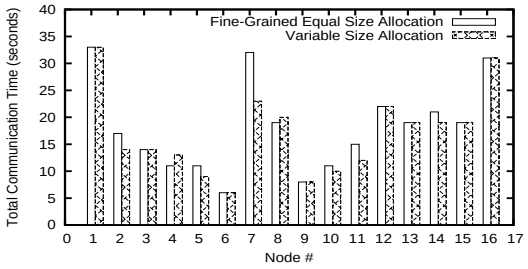


Figure 13: Comparison of Communication Time of Equal Size Fine-Grained vs. Variable Size using Historical Workunit Distribution - DB size 119MB Grid Size 16

Figure 14 shows the complete distribution of 52 workunits among the 16 worker nodes for DB of total size 284 MB. Here too we observe that the first workunit takes more amount of time than the future workunits.

Figure 15 shows the comparison of the total communication time for the equal-size fine-grained allocation case to the communication time in the variable-size allocation case for DB of size 284 MB and Grid size 16. Here we observe that the communication time for some of the nodes in the variable-size allocation is more than that in fine-grained allocation case. This added communication time adds to the total time which is reflected in the results observed in Table 1.

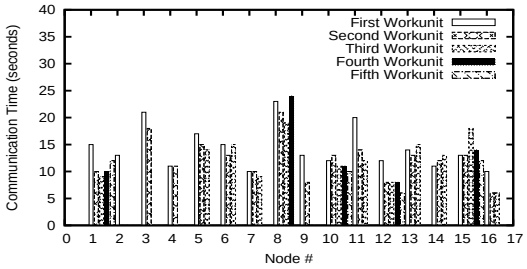


Figure 14: Effect of contention at the server for DB size 284 MB and Grid size 16

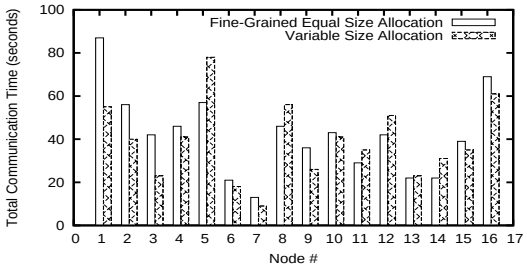


Figure 15: Comparison of Communication Time of Equal Size Fine-Grained vs. Variable Size using Historical Workunit Distribution - DB size 284MB Grid Size 16

4.5 Effect of Placing the Server on the Donation Grid

In the evaluation results presented till now The BOINC server was placed on a dedicated machine that was outside the Planetlab infrastructure. Here we evaluate the effect of placing the BOINC server itself on one of the donation Grid nodes. We evaluate this setup for a database of size 119 MB and Grid of 16 nodes.

Figure 16 gives the effect of varying the number of workunits on the total response time along with the breakdown costs. The optimum number of workunits for this configuration is 40. We see from the results in this figure that the heuristic is obeyed when the server is running on Planetlab.

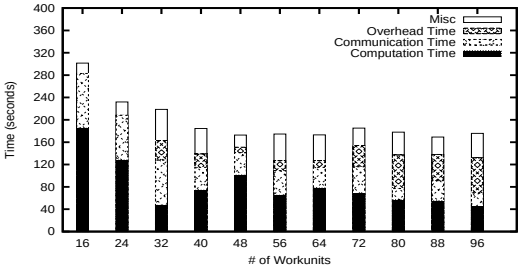


Figure 16: Server on PlanetLab - Effect of varying number of workunits - DB size 119 MB, Grid size 16

Figure 17 gives a comparison of the Equal-Size and the Variable Size workload distribution strategies. Here too we observe that the Fine-Grained workload distribution strategy does better than the

Variable-Size workload distribution strategies. One of the points to be observed here is that the variable-size workload distribution strategies do much worse here than for the same database and Grid size with the server on a dedicated machine. The reason for this can be explained from the Figures 18 and 19.

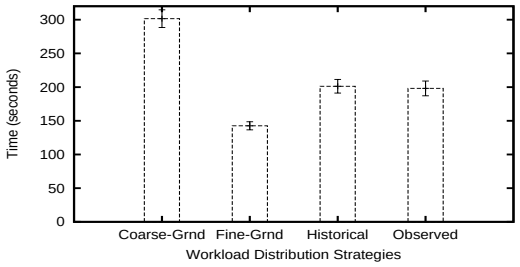


Figure 17: Server of PlanetLab - Equal Size vs. Variable Size Workload Distribution Strategies - DB size 119 MB, Grid size 16

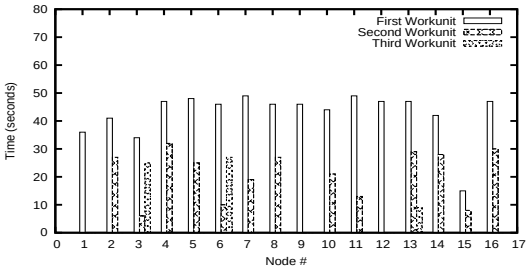


Figure 18: Effect of contention at the server with server on Planetlab

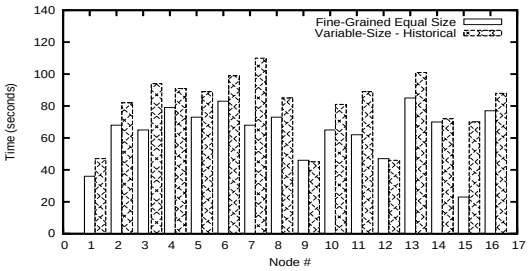


Figure 19: Comparison of Communication Time of Equal Size Fine-Grained vs. Variable Size using Historical Workunit Distribution with server on Planetlab

Figure 18 shows the distribution of all 32 workunits with the server on Planetlab. We see from this figure that the first workunit takes considerably longer amount of time to download than the future workunits. In Figure 19 we observe from the comparison of the sum of the communication time in the fine-grained workunit allocation case to the variable size allocation case that the communication time for all the nodes is greater for the variable size allocation case. This is because with the server running on a Planetlab node there is much greater data contention as the Planetlab node is a shared resource with the bandwidth being shared among the different services running on that node.

Overall looking at the comparison of Equal Size and Variable Size workload allocation strategies in sections 4.4 and 4.5 we observe that the Fine-Grained Equal Size workload allocation does better than the Variable-Size allocation when the contention at the data server is large. When the data server is placed on a dedicated node or where the data contention is not a bottleneck the Variable-Size allocation does almost equally as good or in some cases better than the Fine-Grained Equal Size allocation scheme. As the intra-node variation in capability is observed to be much lesser than the inter-node variation the Variable Size allocation is a useful workload allocation strategy. The decision of choosing a workload distribution strategy could be dynamically done on the basis of the current status of the Grid. If the contention at the data server is not a bottleneck the Grid middleware would choose the Variable Size workload allocation scheme. When the data contention is significant the Fine-Grained Equal Size workload allocation would be chosen due to its better performance in such a Grid environment. Another scenario where the Fine-Grained Equal Size workload allocation would be preferred is when the node churn in the Grid is high.

5. SUMMARY AND FUTURE WORK

In the previous sections we have evaluated the different workload distribution strategies for dealing with the heterogeneity on nodes in a donation-based Grid. In this section, we summarize the results presented in section 4. On the basis of the evaluation results presented in the previous section we make the following conclusions:

- The baseline BOINC workload distribution is not a good approach when we want to run BOINC in an environment where performance is critical as its workload distribution strategy ignores worker heterogeneity.
- The heterogeneity among the worker nodes can be exploited in a number of ways such as by dividing a task into smaller tasks or also by assigning tasks to the worker nodes on the basis of their capabilities.
- We see that the Fine-Grained Equal-Size workunit allocation strategy performs the best in most of the scenarios. This result is useful because this workunit allocation strategy better suits a Grid platform which is donation-based. This is because in a donation based grid there could be node churn with nodes leaving and joining the Grid arbitrarily. The Variable-Size workunit allocation strategy would require the system to monitor the Grid nodes and recalculate the variable-size workunits when the nodes in the Grid change. This adds extra overhead. We have observed such node churn on the Planetlab infrastructure where nodes become inaccessible or go down for certain periods of time. In such an environment the Fine-Grained workunit allocation strategy would work best with minimum overhead and would adapt to the changing environment better. Also the variable size allocation strategy requires intrusive changes to the BOINC scheduler.
- In case of the Fine-Grained Equal-Size workunit allocation strategy we have observed that dividing a task into subtasks works only upto a certain point where the system becomes load-balanced. A further decomposition of tasks does not improve the performance instead the overheads begin the dominate which leads to a drop in the performance.
- A single data server has a significant impact in such a service-oriented BOINC environment when the worker nodes are polling the server constantly. When work is generated at the

server a cascading effect is observed where all worker nodes attempt to communicate and download from the server at the same time which affects the performance of all the work requests present on the server.

The summary points described above form the basis of our future work. We are currently working on a variable size workload distribution strategy that along with sizing the workunits also *staggers* them such that not all workunit input files are downloaded at the same time. In this allocation strategy the faster nodes in the Grid would be given larger chunks of the work initially and would be the first ones to receive the workunits.

Another way of dealing with the data contention at the server is to have multiple data servers. BOINC allows a project to have multiple data servers and the latest BOINC provides mechanisms by which worker nodes download input files from the data server node that falls in the same *timezone*. There are obvious limitations of this approach when we consider a scenario where a majority of the worker nodes lie in the same timezone. This would excessively load the data server in that timezone while the other data servers remain lightly loaded.

One of the other research directions that we are currently working on is smart partitioning of the Grid resources among requests so as to attain best performance for all pending requests as giving all resources to a certain request may actually reduce the performance as a result of the contention.

6. REFERENCES

[1] B. Chun D. Culler S. Karlin S. Muir L. Peterson T. Roscoe T. Spalink A. Bavier, M. Bowman and M. Wawrzoniak. Operating System Support for Planetary-Scale Network Services. In *Proceedings of the Fifth Symposium on Networked Systems Design and Implementation (NSDI'04)*.

[2] D. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004)*, November.

[3] BLAST. http://www.ncbi.nlm.nih.gov/blast.

[4] LHC@Home. http://lhcathome.cern.ch.

[5] Climate Prediction. http://climateprediction.net.

[6] Lattice Project. http://lattice.umiacs.umd.edu.

[7] SETI@Home. http://setiathome.ssl.berkeley.edu.

[8] WorldCommunity. http://www.worldcommunitygrid.org.