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

In this paper, we explore the tradeoffs and opportunities in porting a high-throughput Grid computing middleware to a high-performance service oriented environment. We present the limitations of the Grid computing middleware when operating in such a performance sensitive environment and suggest ways of overcoming these limitations. We focus on exploiting the computation and communication heterogeneity of the Grid resources to meet the performance requirements of services, and present several approaches of work distribution that deal with this heterogeneity. We also present a heuristic for finding the best decomposition of work and present algorithms for each of the approaches which we evaluate on a PlanetLab testbed. The results validate the heuristic and 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 selecting a work distribution policy based on the dynamic status of the Grid computing environment.

1 Introduction

Grids that employ donated resources have become an effective means of performing large-scale computations. One of the first projects that made use of a donation-based Grid was the SETI@home project [19]. Donation-based Grids have now found application in a diverse set of domains such as Physics [14], Weather Forecasting [17] and Medical Research [22]. 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. 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 similar to 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 [18]. In a service-oriented environment, a service request 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 measured by its response time for individual service requests.

Since requests for large Grid-based services would typically consist of multiple tasks, the service performance for each request becomes inherently dependent on the relative performance of individual tasks. A request finishes only when its last task finishes, the so-called *makespan* of the request. Thus, a request's performance can be improved mainly by reducing its makespan, i.e., by minimizing the completion time of the last task. This raises an important challenge when hosting such a service on a heterogeneous set of resources. If the tasks are not distributed properly, it is possible for a slow node to become the bottleneck for the entire request even if the other tasks finish quickly. This observation suggests that any workload

allocation strategy that distributes tasks to Grid resources must incorporate heterogeneity information into its decisions.

In this paper, we explore several workload distribution strategies that exploit the Grid resource heterogeneity to make better scheduling decisions.The aim of these workload distribution strategies is to distribute the workload in proportion 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 such that the faster nodes in the Grid perform more work. We then propose strategies that make use of observed performance to estimate the capabilities of worker nodes for the creation and assignment of matching tasks to them. We evaluate these workload distribution strategies by implementing them in the BOINC [2] middleware running on PlanetLab [7], a planetary-scale distributed testbed. We have used BLAST [10], 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 present experimental results that motivate the problem. In Section 3, we propose different workload distribution strategies to exploit the heterogeneity of the resources. Section 4 presents a performance evaluation and comparison of the different strategies. We presented related work in Section 5 and summarize our results in Section 6.

2 Donation-based Grids and Heterogeneity

We begin by describing the typical donation-based Grid model represented by BOINC: a popular middleware used for hosting projects such as SETI@home [19] and Climate prediction [17]. We then describe our experimental framework and present experimental results illustrating the challenge of significant heterogeneity in such an environment. These results provide a motivation for the problem we address in this paper.

2.1 System Model and Experimental Framework

The BOINC system consists of single centralized scheduler which consists of two major components. A scheduling server which hands out 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 uses a pull-based scheduling model where the worker nodes poll the BOINC server periodically, requesting work. The worker nodes send back the output files to the server after completing the computation. All tasks are independent and require no interaction between the worker nodes. Hence the only communication is between the server and the worker nodes. We use BOINC as the middleware for implementing our workload allocation

In our experimental framework, we executed BOINC on the PlanetLab infrastructure: a shared distributed infrastructure consisting of donated machines. In our testbed, the BOINC worker nodes were a set of 16 randomly selected PlanetLab nodes, each running the Fedora Core 2 Linux kernel 2.6.8. The nodes had varying hardware capabilities and were geographically distributed. Most of the worker nodes were Pentium III or Pentium 4 machines with CPU speeds ranging from 1.2 GHz to 3.0 GHz, memory ranging from 1GB to 2GB, and each having about 5GB of disk space. The BOINC Grid server ran on a dedicated machine that is outside the PlanetLab infrastructure. We used the BOINC development version 4.72 to set up our Grid prototype on the PlanetLab testbed.

We modified a standalone bioinformatics application called BLAST (Basic Local Alignment Search Tool) to run as a Grid service on BOINC. 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

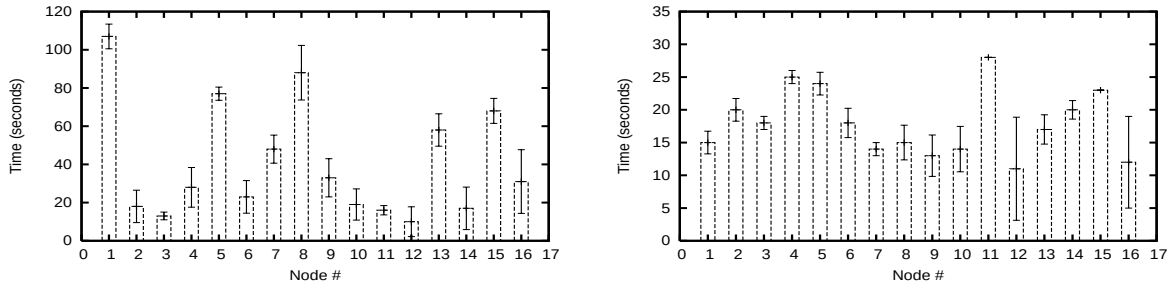
compares it to a formatted database file and generates an output file containing a similarity score and similarity matches with the database. The BLAST application serves as a good representative for a Grid service as it is both computationally intensive and data-rich, requiring the transfer of a large amount of data to perform the computation. In our experiments, we used a 119 MB (drosoph.nt) and a 284 MB (sts) formatted file of sequences as the BLAST databases of gene sequences. The input sequence used for comparison was a randomly-selected sequence from the database; the sequence was of length 569 bytes.

Next, we present results from a set of experiments in our testbed that demonstrate the presence of significant heterogeneity in a donation-based Grid environment.

2.2 Demonstration of Grid Heterogeneity

In our first set of experiments, we used the 119 MB BLAST database for gene comparison. Each of the 16 worker nodes was given an equal share of the work by splitting the database into 16 equal-sized chunks. In each run, the database chunks were transferred from the BOINC server to the worker nodes, and the results returned to the server after the computations. For each run of our experiments, we measured the total request execution times along with the component costs such as computation and communication times at each worker node. The communication time is largely dominated by the transfer of the database chunk as the input sequence file and the resultant output files are comparatively much smaller in size.

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. Figure 1(b) shows similar results for the communication time, indicating the disparity in the bandwidths or the link speeds of the different nodes from the BOINC server. The degree of heterogeneity observed for communication does not appear to be as pronounced as that for computation. We conjecture that this observation is due to the generally strong connectivity of PlanetLab nodes (most of which are on Internet2), and expect communication heterogeneity to be much more pronounced for typical Grids where many nodes may be behind slow "last-mile" connections.



(a) Computation Time (b) Communication Time

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

Another interesting observation we make from Figure 1(a) is the difference between the inter-node vs. the intra-node variation in computation time. The values of the bars in the figure indicate the presence of large inter-node variation in computation time, while the tight standard deviations imply small intra-node variation even across multiple runs. A similar observation can be made about communication times from Figure 1(b). These observations suggest that there are consistent differences in the capabilities of different nodes, which are distinguishable across time. Moreover, the per-node variations are small enough that they would not

impact the inter-node differences significantly. This makes it possible to exploit Grid heterogeneity without having to worry about dynamic intra-node variations.

Having demonstrated the presence of significant heterogeneity across Grid resource, we now present work allocation techniques that can exploit such differences to improve the performance of Grid services.

3 Workload Distribution Strategies

As described in Section 2, we observed that the nodes in a typical donation-based Grid exhibit substantial heterogeneity due to their difference in compute capabilities and network bandwidth. 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. We now describe workload distribution strategies that could be applied to the BOINC Grid infrastructure.

3.1 Homogeneous Workunit Allocation

Homogeneous workunit allocation strategies create workunits of equal size which can therefore be allocated to any worker. We first define a simple heuristic that creates a number of coarse-grain workunits equal to the number of nodes in the grid. This is the scheme used in Section 2, and it serves as a point of comparison. For this reason, it is our baseline. It is easy to implement since it is clear how to create workunits. The other approach is a fine-grained equal-size workload allocation, FG-ES, that creates a larger number of smaller workunits.

3.1.1 FG-ES

FG-ES addresses the heterogeneity of worker nodes by subdividing the workunits into finer chunks. This strategy leads to a better workload distribution because faster nodes pick up more work and better load balance is achieved. The main idea behind FG-ES is to load balance the Grid by creating finer units of work. However, there is a point of diminishing returns, beyond which the overheads of server-worker interactions would outweigh the benefits of load balancing. The question is how many workunits should a given amount of work be decomposed into in order to achieve the "best"[1] decomposition for a given problem.

We present a heuristic to determine the best decomposition of work for a given problem size (Algorithm 1). It is guided by the observation that as the work is decomposed into finer units of work, an improvement 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, and would depend upon the relative cost of computation and communication for the application. The point where the difference in the granularity drops below this constant δ is the best decomposition for the current problem.

We illustrate the heuristic by applying it to the BLAST application used in our experiments, given a particular database and Grid size. In particular, for BLAST, we have: $Problem_{size} = Database_{size}$ and $Init_{decomp} = Grid_{size}$. 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*.

[1] By best, we mean it achieves the best results experimentally over the range of configurations we explored

Algorithm 1 Workload-Decomposition(*$Problem_{size}$*, *$Init_{decomp}$*, *step*, δ)

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

The value of δ depends on the step size, which we denote as δ_{step}. When the difference in the size of the database chunk is less than δ_{step} a further improvement in the total time will not be observed.

For BLAST, the relation between step and δ_{step} was empirically determined to be:

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

The step value was chosen to be 4 due to the nature of the BLAST databases. The database file consists of a list of gene sequences and it is not possible to arbitrarily split the database since the database chunks have to be aligned to the start and end of complete sequences. A minimum step size of 4 is chosen to account for this alignment error. The choice of this step value in general would depend on the application, Grid environment, and efficiency considerations (a smaller step leads to a greater number of decompositions to be explored at runtime).

3.2 Heterogeneous Workunit Allocation

The FG-ES strategy requires that the worker nodes interact with the server frequently to acquire additional workunits. Heterogeneous workunit allocation strategies attempt to eliminate this overhead by creating a smaller number of variable sized workunits by matching the size of a workunit to the relative capability of the node. The server does selective scheduling by having a worker node pick up a specific workunit according to its capability. This workunit allocation strategy requires a method of estimating the capabilities of nodes in order to create these different-sized workunits. We have developed three approaches for variable-size allocation:

- VS-benchmark: this uses benchmark information collected by BOINC
- VS-dynamic: this uses prior runs of the baseline to determine computation and communication rates of each node
- VS-history: this uses the historical information of workunit distribution observed in prior runs of FG-ES

3.2.1 VS-benchmark

The BOINC core client collects benchmark information when it 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, 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.

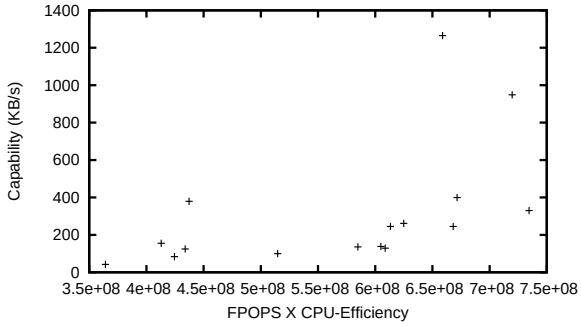


Figure 2: Compute Capability as a function of Fpops and CPU-efficiency

In Figure 2, we plot the observed compute capability of 16 Grid nodes as a function of their Fpops and CPU-efficiency values. The compute capability is expressed in units of KB/s to capture the rate of consumption of BLAST data. We observe from the figure that the data points are widely scattered. 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. These results suggest that the BOINC-collected benchmark information is insufficient to estimate the capability of a Grid node. Also these benchmarks do not capture the nature of the BLAST application which is data oriented and hence depends on the amount of available memory and I/O capability of the node. This difference in the nodes make the BOINC benchmarks unsuitable for estimating a node's capability in executing a BLAST request. Next, we explore two approaches to heterogeneous workload allocation that make use of observed performance of Grid nodes to better estimate the node capabilities.

3.2.2 VS-dynamic

VS-dynamic uses the observed node computation and communication rates to decide the size of the workunits to be allocated to nodes in the Grid. This strategy collects the computation and communication rates for each node (CNC[node]) from prior runs of the baseline strategy, coarse-grained homogeneous workunit allocation. This value is used to get an estimate of the node capability. The Proportion$_{factor}$ gives an estimate of the amount of time the Grid should take to compute a request with respect to the entire database, based on the total capability of the Grid. The capabilities of all Grid nodes are then used to compute the proportional share of the database to be assigned to each node such that the Grid nodes finish their computation at approximately the same time (Algorithm 2).

Algorithm 2 Var-size-Dynamic (*DB_{size}*, *$Grid_{size}$*, *Nodes[]*, *CNC[]*)

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 VS-history

VS-history uses information obtained from prior runs of FG-ES to compute the size of the variable size workunits. The FG-ES strategy subdivides the workunits into finer chunks based on the heuristic method described in section 3.1.1 that finds the best number of workunits for a particular database size and grid size. Instead of having the worker nodes fetch additional workunits from the server, the entire workload for a node (a chunk of the database) is assigned to it once. The function Group-Database-Chunks() groups as many database chunks as computed by the node in the FG-ES case into a single chunk which forms the variable size chunk for that node (Algorithm 3) .

Algorithm 3 Var-size-Historical (*DB_{size}*, *$Grid_{size}$*, *Nodes[]*)

1: Best$_{workunits}$$\leftarrow$FG-ES(DB$_{size}$, Grid$_{size}$)
2: Obtain Response$_{times}$[] with FG-ES for Best$_{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 3.1.1 and evaluate the performance of the different workload distribution strategies presented in the previous section. Our experimental setup is as described in Section 2.1.

4.1 Baseline vs. FG-ES

In this section, we compare the baseline allocation strategy against FG-ES. From Figure 3, it is clear that a significant improvement in the total response time is possible by creating finer-grained workunits. One other point to note is that the performance improvement is greater when the total size of the database is larger. The reason is that for a larger database size the baseline creates a greater disparity among the worker nodes.

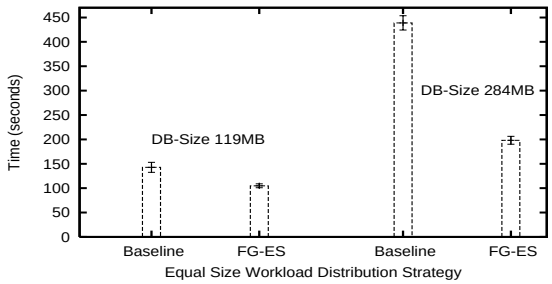


Figure 3: Comparison of Equal-Size Workunit Allocation Strategies for databases of size 119 MB and 284 MB

4.2 FG-ES: Detailed Evaluation

FG-ES achieves better workload distribution by creating finer-grained workunits. Figure 4 illustrates how accurately the worker capabilities correspond to their workunit allocations. Node capability is the combined computation and communication capability of the node expressed in units of KB/s.

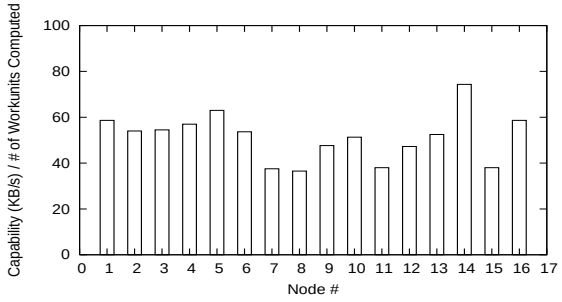


Figure 4: Ratio of node capability to workunit distribution for DB of size 119 MB

From Figure 4 we observe that the ratio of node capability to the number of workunits executed by that node has relatively little variation across nodes when compared with the variance seen for the baseline case (Figure 1(a)). This shows that the fine-grained workunit allocation better load-balances the Grid nodes.

4.2.1 Heuristic Validation

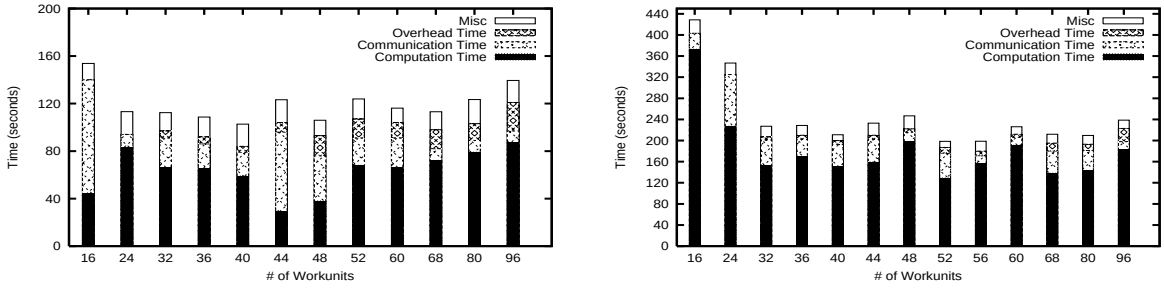
In section 3.1.1 we presented a heuristic which determines the best number of workunits for a given database and Grid size. In this section, we validate this heuristic under different configurations. We also present a breakdown of the total time into computation, communication and overhead component times, to give insight into the heuristics. 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 a node is 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 the time taken by the slowest node, which is the maximum of all the node times. The overhead time will rise as the number of workunits are increased. 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.



(a) DB size 119 MB

(b) DB size 284 MB

Figure 5: Effect of varying the number of workunits

Figures 5(a) and (b) 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 for the slowest node in the Grid, the bottleneck node. 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, hence the computation time is reduced.

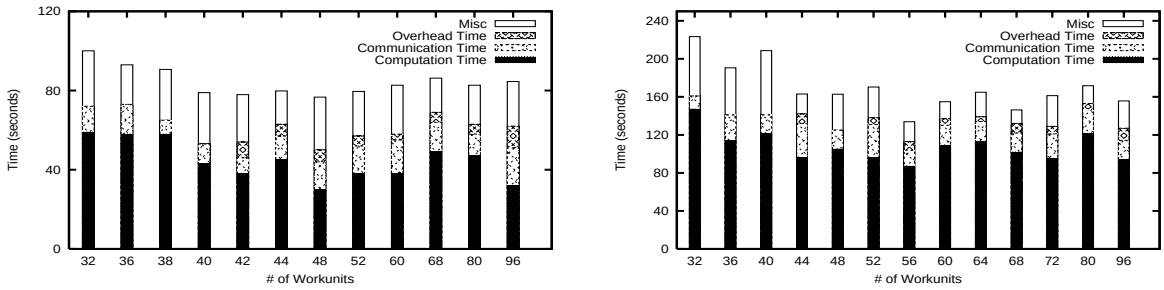
For the database of size 119 MB the best number of workunits is 40 as obtained from the heuristic. We see that the beyond 40, the total response time flattens out and then rises gently. The heuristic stops at the point where the curve starts flattening out indicating no further improvement is possible. The gentle rise in the total time as the number of workunits increases is due to the overhead of dispatching more workunits. Figure 5(b) shows the effect of varying number of workunits for a database of size 284 MB. For this database size the best number of workunits is 56, which is again returned by our heuristic.

4.2.2 Effect of Increasing Grid Size

In this section, we study the effect of increasing the Grid size. Figures 6(a) and (b) show the effect of varying the number of workunits for a database of size 119 MB and 284 MB for a larger Grid of size 32.

For the database of size 119 MB, the best number of workunits is 40 and for the database of size 284 MB, the best number is 56. The first point to observe is that a larger Grid reduces the benefit of FG-ES vs. the baseline. The reason is that with a larger Grid size, the size of each database chunk for the baseline is small. Hence the heterogeneity of the nodes is less pronounced. Comparing Figures 5(b) and 6(b), 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 it is about 77 seconds.

We also observe that as we increase the number of workunits, the overheads rise more sharply with a larger Grid. The reason is that there are more 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.



(a) DB size 119 MB (b) DB size 284 MB

Figure 6: Effect of varying the number of workunits for Grid size 32

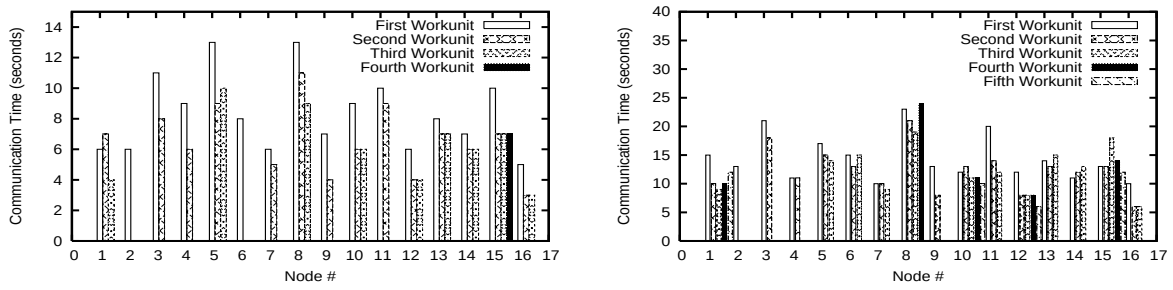
		Homogeneous (secs)		Heterogeneous (secs)	
DB-size	Grid-size	Baseline	FG-ES	VS-history	VS-dynamic
119MB	16	153	102	102	96
284MB	16	428	198	227	244
119MB	32	100	78	84	90
284MB	32	223	133	168	146

Table 1: Comparison of the homogeneous and heterogeneous workload allocation strategies

4.3 Comparison of Homogeneous and Heterogeneous Strategies

In this section, we compare the homogeneous (Baseline and FG-ES) workload allocation strategies with the heterogeneous (VS-history and VS-dynamic) strategies. As we have seen, there is an overhead associated with a worker node interacting with the server. The intuition behind the variable-size workunit allocation strategies is to reduce the overhead by grouping workunits based on node capability.

Table 1 compares the workload distribution strategies for different database sizes and Grid sizes. We observe that the results obtained are counter-intuitive as for 3 out of the 4 cases, FG-ES does better than the variable-size allocation strategies. The only case in which the variable-size allocation strategy performs the same as or better than the FG-ES allocation strategy is for the database of size 119 MB and Grid size of 16. This is because the effect of contention at the server is smallest and the VS-* strategies do better.



(a) DB size 119 MB (b) DB size 284 MB

Figure 7: Effect of contention at the server for Grid size 16

The effect of contention can be observed from Figure 7(a) which shows the complete distribution of 40 workunits among the 16 worker nodes for a DB size of 119 MB. We see that the first workunit at each node

takes more time to download than the subsequent workunits. This is because contention at the server is highest for the first workunit as all the worker nodes download the input files at the same time. Figure 7(b) 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 longer time than subsequent workunits. For all other cases, increasing the DB size to 284 MB or increasing the Grid size to 32 increases the contention at the server which increases total time, and therefore FG-ES performs better than VS-* strategies.

4.4 Effect of Placing the Server on the Donation Grid

In the results presented thus far, 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 on one of the PlanetLab donation Grid nodes. We evaluate this setup for a database of size 119 MB and Grid of 16 nodes.

Figure 8 shows the effect of varying the number of workunits on the total response time along with the breakdown costs. The best number of workunits for this configuration is still 40 as predicted by the heuristic.

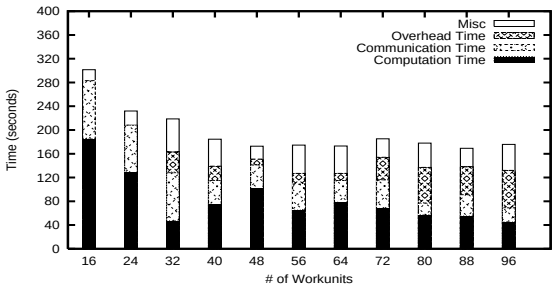


Figure 8: Server on PlanetLab - Effect of varying number of workunits - DB size 119 MB, Grid size 16

Figure 9 gives a comparison of FG-ES and VS-* workload distribution strategies. Here too we observe that the FG-ES does better than the VS-* strategies. However, VS-* does far worse for the same database and Grid size compared with the server on an outside machine. The reason for this can be explained from Figures 10 and 11.

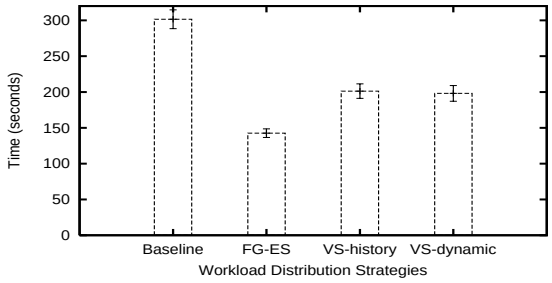


Figure 9: Server on PlanetLab - FG-ES vs. VS-* - DB size 119 MB, Grid size 16

Figure 10 shows the distribution of all 32 workunits with the server on PlanetLab. We see from this figure that the first workunit takes considerably longer to download than subsequent workunits. In Figure 11 we observe that the cumulative communication time is smaller for FG-ES as compared to VS-*. This is because the PlanetLab server shares its bandwidth with different services running on that node.

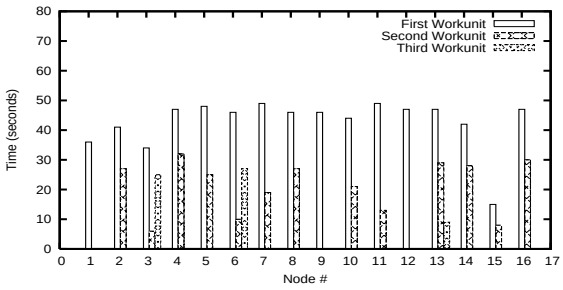


Figure 10: Effect of contention at the server with server on PlanetLab

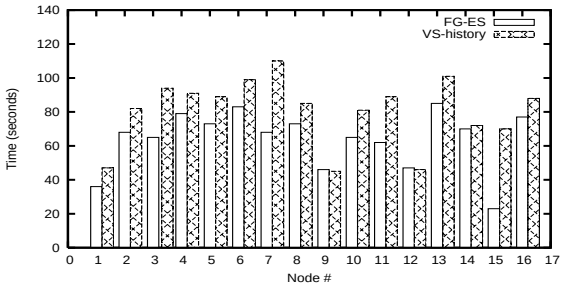


Figure 11: Comparison of Communication Time of FG-ES vs. VS-history with server on PlanetLab

In summary, we observe that FG-ES outperforms VS-* when 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, VS-* is almost as good and in some cases better than FG-ES. The best workload distribution strategy depends on the current dynamics of the Grid and server placement. Another scenario where FG-ES would be preferred is when the node churn in the Grid is high since a smaller investment is made if a node executing a smaller workunit leaves. Also FG-ES handles dynamicity in load better: if a node gets heavily loaded during the execution of a request, it would effectively compute fewer workunits. In contrast, the VS-dynamic and the VS-history strategies are interleaved with the baseline and the FG-ES strategies respectively, and hence assume the load to be constant during the execution of a request.

We are also looking at ways of improving these workload distribution strategies. The issue of contention at the server can be managed by staggering workunits at the data server such that there is minimum download overlap. The FG-ES strategy can be improved by overlapping computation of a workunit with the communication for the next workunit. This would help further reduce the total makespan.

5 Related Work

A substantial body of work on load-balancing has focused on cluster based distributed systems. Systems such as Condor [15] and Mosix [6] rely on checkpointing and process migration to load balance a cluster of workstations. The heterogeneity of the cluster of workstations is managed by dynamically collecting load information and migrating actively running processes between cluster nodes to balance the load. Such load-balancing techniques can be complimentary to our work allocation techniques that focus on initial allocation of tasks according to node capabilities.

Clusters of workstations have also been employed to host Web and Internet servers. A large body of work on such cluster-based network servers has focused on request distribution as a means for handling

the load imbalance in the cluster. Load-aware request distribution [16] and [4] use content-based request distribution which take into account the locality of data and the load on the cluster nodes. Aron et al. [3] focus on request isolation and resource management on cluster based network servers while [20] proposes cluster load balancing policies for fine grain network services.

Load sharing in heterogeneous systems has been widely researched. [5] evaluates different load sharing algorithms for heterogeneous multicomputer systems. Goswami et al. [11] propose dynamic load sharing heuristics which manage workload in a distributed system by predicting the resource requirements of processes. Concert [1] uses a proactive load sharing scheme for loosely coupled distributed systems which avoids the occurrence of load imbalance by collecting load and task behavior information in advance. Karatza et al. [12] investigate load sharing policies for heterogeneous distributed systems to study the effect of load sharing on different classes of jobs. Berman et al. [9] describe an application specific scheduling approach for scheduling data parallel applications on a heterogeneous distributed system. Nieuwpoort et al. [21] describe load balancing strategies specifically for divide and conquer applications on a hierarchical wide-area distributed system. Kondo et al. [13] consider a similar system model as ours and propose techniques for resource selection for short-lived applications on enterprise desktop Grids with the aim of minimizing the overall execution elapsed time of a single parallel application. We consider a similar scenario but propose algorithms and heuristics for deciding the decomposition of tasks in order to load balance a heterogeneous set of Grid resources. Such scheduling algorithms have also been an active area of research in the field of divisible load scheduling. [8] provides an overview of the research done in this field for master/worker architectures.

6 Conclusions and Future Work

The presence of computation and communication resource heterogeneity in a donation-based Grid is an obstacle for performance-sensitive services-oriented applications. We proposed various workload distribution strategies to handle the Grid heterogeneity that improve the performance of request-oriented Grid services. Our evaluation, conducted on the BOINC middleware running on the PlanetLab infrastructure, showed that a fine-grained work allocation strategy that allows nodes to execute multiple tasks according to their capacity, performs well in large number of scenarios. We also evaluated variable-size strategies, one of which used historical information from the fine-grained strategy, and another used dynamic node capability information. Our results demonstrated that these strategies performed better than the baseline strategy, however, they suffered from increased contention at the data server, though this contention may be reduced in the presence of a high-bandwidth server. Using BLAST as an example service, we demonstrated how the fine-grained workunit allocation strategy is more suitable for data-rich services due to the issue of contention at the data server. In order to extend our work to other data-rich services we would only need to find the appropriate threshold for "best" decomposition. We intent to model this as a function of the computation and communication time taken for a service request. Overall our results demonstrated that incorporating heterogeneity information into workload allocation decisions can substantially improve the performance of services on donation-based Grid environments.

As part of future work, we intent to extend our work to an environment with multiple services and concurrent requests. We will explore the impact of the workload allocation choices on the performance of concurrent requests and investigate the feasibility of providing service differentiation across requests, request classes, or services.

References

[1] R. Anane and R. J. Anthony. Implementation of a Proactive Load Sharing Scheme. In *Proceedings of the 2003 ACM symposium on Applied computing*, 2003.

[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)*.

[3] Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Cluster Reserves: A mechanism for Resource Management in Cluster-based Network Servers. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 2000.

[4] Mohit Aron, Darrel Sanders, Peter Druschel, and Willy Zwaenepoel. Scalable Content Aware Request Distribution in Cluster-Based Network Servers. In *Proceedings of the USENIX 2000 Annual Technical Conference, San Diego, CA*, 2000.

[5] Sayed A. Banawan and Nidal M. Zeidat. A comparative study of load sharing in heterogeneous multicomputer systems. In *Proceedings of the 25th Annual symposium on Simulation*, 1992.

[6] A. Barak, S. Guday, and Wheeler R. The MOSIX Distributed Operating System: Load balancing for Unix. *Lecture Notes in Computer Science, Springer-Verlag*, 672, 1993.

[7] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, 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)*.

[8] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling Divisible Loads on Star and Tree Networks:Results and Open Problems. *IEEE Transactions on Parallel and Distributed Systems (TPDS), 16(3), 207–218*, 2005.

[9] Fran Berman, Rich Wolski, Silvia Figueira, Jennifer Schopf, and Gary Shao. Application Level Scheduling on Distributed Heterogeneous Networks. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing*, 1996.

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

[11] K. K. Goswami, M. Devarakonda, and R. K. Iyer. Prediction based Dynamic Load-Sharing Heuristics. In *IEEE Transactions on Parallel and Distributed Systems*, 1993.

[12] Helen D. Karatza and Ralph C. Hilzer. Load Sharing in Heterogeneous Distributed Systems. In *Proceedings of the 2002 Winter Simulation Conference*, 2002.

[13] Derrick Kondo, Andrew A. Chien, and Henri Casanova. Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids . In *Proceedings of the 2004 ACM/IEEE conference in Supercomputing*, 2004.

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

[15] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems(ICDCS)*, 1988.

[16] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich Nahum. Locality Aware Request Distribution in Cluster-Based Network Servers. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1998.

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

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

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

[20] Kai Shen, Tao Yang, and Lingkun Chu. Cluster Load Balancing for Fine-Grain Network Services. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, 2002.

[21] Rob V. van Nieuwpoort, Thilo Kielmann, and Henri E. Bal. Efficient Load Balancing for Wide-Area Divide-and-Conquer Applications. In *Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, 2001.

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