Grid-Enabling a Vibroacoustic Analysis Application

Brian Bentow
Jon Dodge
Aaron Homer
Christopher D. Moore
Robert M. Keller

Harvey Mudd College
Claremont, CA

bbentow@cs.hmc.edu
jdodge@cs.hmc.edu
ahomer@cs.hmc.edu
cdmoore@cs.hmc.edu
keller@cs.hmc.edu

Matthew Presley
Robert Davis
Jorge Seidel
Craig Lee
Joseph Betser

The Aerospace Corporation
El Segundo, CA

presley@aero.org
robdavis@aero.org
seidel@aero.org
lee@aero.org
Joseph.Betser@aero.org

***Abstract*— This paper describes the process of grid-enabling a vibroacoustic analysis application using the Globus Toolkit 3.2.1. This is the first step in a project intended to grid-enable a suite of tools being developed as a service-oriented architecture for spacecraft telemetry analysis. Many of the applications in the suite are compute intensive and would benefit from significantly improved performance. In this paper we show the advantage of using Globus to grid-enable a single tool in a vibroacoustic analysis flow, with the result that using as few as eleven nodes, that tool's runtime improved by a factor of eight. While communication overhead does affect performance, these results also indicate that coordinated communication and execution scheduling as part of workflow management would be able to significantly improve overall efficiency. In the larger context, our experience also shows that the service-oriented architecture approach, using grid computing tools, can provide a more flexible system design, in addition to improved performance and increased utilization of resources. We also provide some lessons learned in using the Globus Toolkit.**

I. Introduction

Space launches are costly, high-risk, multi-discipline endeavors, and mission payloads must be carefully engineered to ensure success. Simply surviving the launch is an important hurdle since satellite payloads are subjected to intense mechanical vibration and acoustic noise. In order to monitor, understand, and better predict this environment, The Aerospace Corporation is developing a Java-based tool suite called Vibroacoustic Intelligent System for Predicting Environments, Risk, and Specifications (VISPERS) that has tools to allow analysts to clean-up, filter and analyze vibroacoustic data [1]. Certain components of VISPERS are compute-intensive, however, and performance could be improved by parallelizing those applications and running them on whatever resources are available. It would also be very useful to allow multiple telemetry analysts to use the tools in an interactive fashion and process data from different launches simultaneously. Hence, it was decided to investigate the implementation of VISPERS as a grid-enabled *service-oriented architecture* to accomplish these goals.

This paper reports on the first sub-task, which is to determine the feasibility of parallel grid-enabling one component of VISPERS, a utility called VAIL, using the Globus Toolkit 3.2.1 [2] in order to achieve a significant performance enhancement. The grid-enabled version is called gVAIL.

II. Motivation

A. Telemetry Collection

When a spacecraft is launched it contains many sensors, including accelerometers to measure vibration and microphones to measure acoustic noise. The data provided by these sensors is converted from analog to digital, then consolidated into a single stream of digital data and sent by radio to telemetry data receiving stations (TDRS) along the path of the spacecraft, as illustrated in Figure 1. This telemetry stream will be picked up by the nearest receiving station, but will likely include static and other anomalies. For example, the signal may briefly disappear (drop out) due to the antenna on the spacecraft rotating away from the receiving station.

The data that VISPERS is used to analyze is provided by all of the telemetry stations along the flight path, both fixed stations and temporary stations on aircraft or ships. The multiple telemetry streams are placed in files in a telemetry database. Once there, they need to be scrubbed for noise, and consolidated into a single, best data stream prior to final processing. It is possible that hundreds of telemetry streams may have to consolidated and analyzed to identify episodes of excessive vibration during launch.

This sequence of steps is illustrated in Figure 2. From the desktop, an analyst can select telemetry streams from the database front-end, construct probability density functions (PDFs) for later comparisons, and use gTACT (Telemetry Alignment and Consolidation Tool) to combine streams. (gTACT is not

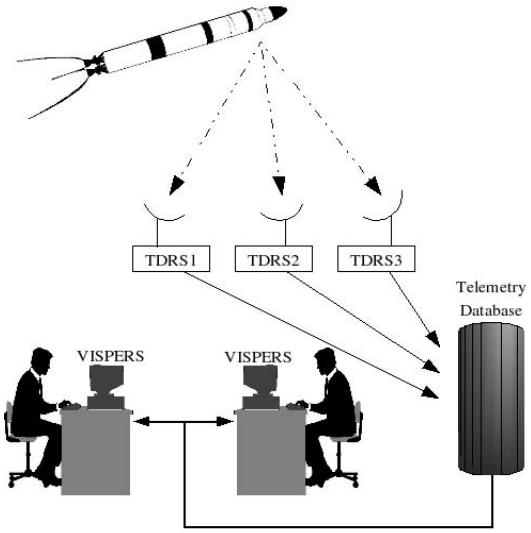


Fig. 1. Telemetry Collection.

discussed here.) Then gVAIL is run to identify possible episodes of interest. Additional tools are available to look for maximum vibrational episodes, analyze damage to the telemetry stream and also edit streams by hand. The ultimate goal is, of course, to help engineers design parts that are unlikely to break in this severe environment.

As part of the VISPERS tool suite, a group at The Aerospace Corporation is looking at using Artificial Intelligence techniques, specifically neural nets, to locate anomalies in the telemetry streams and to suggest to the human analyst ways to fix them. This group has developed a utility called VAIL (VISPERS AI Lab). VAIL looks at the data stream by doing a DSP analysis, and one or more *neurons* fire when an anomaly is detected. If the human analyst agrees on the location and type of anomaly the behavior of those neurons is reinforced as necessary. VAIL then suggests a fix for the anomaly from several possible tools based on the choice the human analyst has made for similar anomalies in the past.

VAIL is extremely compute-intensive. Not only does it look at (do a DSP analysis of) every point in the waveform and the neighborhood around that point, but different neurons look for different anomalies (noise spikes, data drop-out, DC drift or signal bleed-through to name just a few) so it is useful to split VAIL's computation across multiple machines. This task is facilitated by VAIL's highly parallelizable algorithm. Because VAIL's analysis on one part of a file does not depend on its analysis on another part of the file, a task can be divided so that each machine analyzes a separate section of a data file. To leverage the computational power of machines across heterogeneous networks, it is advantageous to employ grid computing, distributed computing that allows computational resources to be shared across disparate networks and organizations [3].

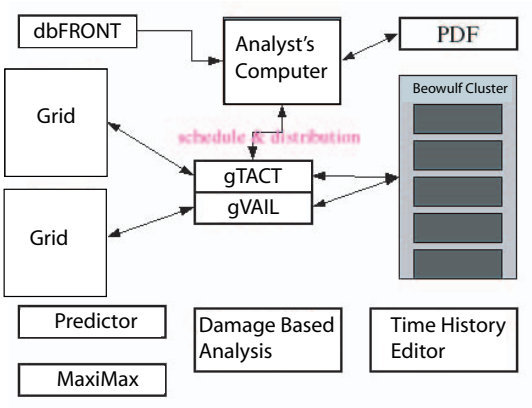


Fig. 2. The VISPERS Workflow.

III. gVAIL Design

A. gVAIL Components

gVAIL consists of the following components: a client program, a scheduler service, an analyzer factory service, a node information publisher service, and Globus's index service [4]. The client program, dubbed the *Time History Editor*, has a GUI that allows a user to view and modify data files (known as *time histories*). It hooks into the rest of gVAIL to allow the user to look for anomalies in the data. The client submits analyses to the scheduler service, which queries Globus's index service to find out which nodes are available. The scheduler (or resource broker) then decides which node should perform which part of the analysis. Currently, it makes this decision using a round-robin algorithm in which each node is assigned an equal amount of work, defined by the number of data points the node is to analyze. Each node runs both the analyzer service, which provides an interface into the VAIL neural network code, as well as the node information publisher service. The node information publisher exposes a service data element (SDE) called `gVAILNodeInfo` and causes the index service to subscribe to this SDE. Currently, `gVAILNodeInfo` has no used data fields. Instead, the scheduler uses the SDE's `originator` attribute to determine the node's IP address and its `goodUntil` attribute to determine whether the node is available. In the future, more information could be published through `gVAILNodeInfo`, such as the node's load, which the scheduler could use to implement a more complicated resource brokering algorithm.

The sequence of steps taken when the user submits an analysis is depicted in Figure 3. The steps are as follows:

1) When the user tells the client to begin an analysis, the client talks to the scheduler, telling it which data file the user wants to analyze.
2) The scheduler queries the index service, asking for a list of machines available to perform analyses.
3) The index service responds with the list of nodes.
4) The scheduler decides which node should perform which part of the analysis and passes this information on to the client.

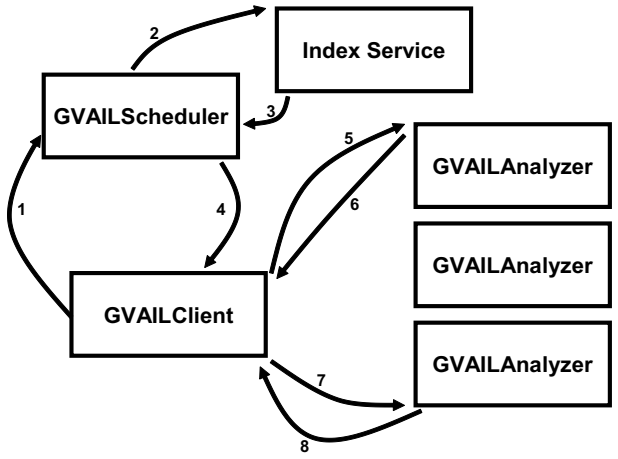


Fig. 3. gVAIL component interactions during an analysis.

5) The client tells the analyzer service on each node to perform its assigned portion of the analysis.
6) Each node spawns off an instance of the analyzer service to perform the analysis, sending status messages to the client to keep it informed of the node's progress.
7) When the node indicates that it has finished, the client requests the results.
8) The node sends the results to the client.

B. gVAIL Code

The gVAIL class structure is depicted in Figure 4. Classes `GVAILBatchModeDriver`, `GVAILBatchRun`, and `GVAILBatchFile` compose gVAILBatchModeClient, a special client designed to perform many consecutive gVAIL analyses (IV-B). gVAILBatchModeClient reads an XML configuration file that describes the analyses to be run. It then executes each analysis sequentially. `GVAILBatchModeDriver`, the interface into the program, uses an instantiation of `GVAILBatchFile` to parse the XML configuration file and then instantiates a `GVAILBatchRun` object to run either a gVAIL analysis or a VAIL analysis (for performance comparison). If a gVAIL analysis is to be run, `GVAILBatchRun` creates a `GVAILTimeHistoryAnalyzer` object, which calls the scheduler's `ScheduleAnalysis()` method to divide the analysis. The scheduler is implemented in `GVAILSchedulerImpl`. Its `ScheduleAnalysis()` method uses an object of type `GVAILQueryHelper` to query the index server for a list of nodes available to perform the analysis. It schedules the analysis and returns a list of `ChunkAssignments` to the client, specifying which node should analyze each region of the file. `GVAILTimeHistoryAnalyzer` then creates a `GVAILAnalyzerTask`, which invokes the `runAnalysis()` method on each analyzer (implemented by `GVAILAnalyzerImpl`). This method creates a thread (`GVAILAnalyzerThread`) to perform the analysis and periodically sends updates in the form of `GVAILServiceObjects` to the client. The

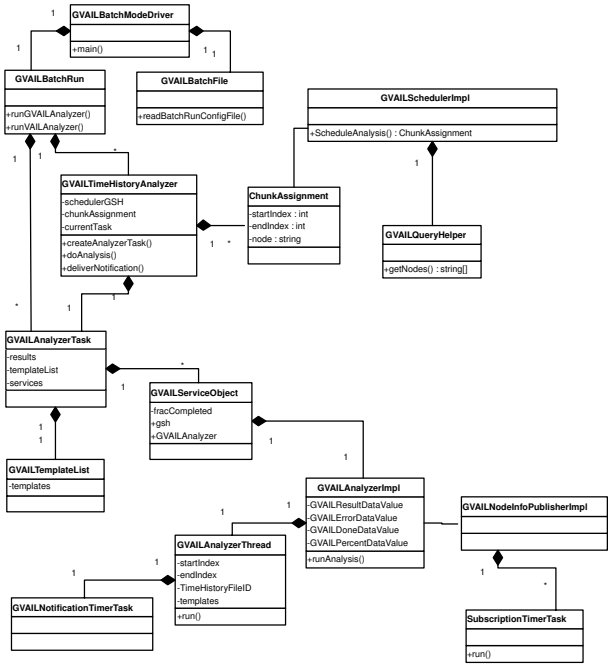


Fig. 4. gVAIL UML Diagram.

node information publisher is implemented in class `GVAILNodeInfoPublisherImpl` and uses a timer (implemented in `SubscriptionTimerTask`) to periodically resubscribe itself to the index service.

C. Service Model vs. Job Model

The primary objective of the project was to determine the feasibility of using the Globus Toolkit to grid-enable applications in a production environment. Consequently, the authors looked at both the job [5] and service models [6] for grid-enabling an application. Eventually, however, the service model was chosen.

The models differ in a number of ways, but most important for this project are their internode communication mechanisms. In the job model, the process sending information outputs it to `stdout`, which is sent to the receiving node. This node must then parse the stream of text. Using grid services, nodes can communicate through either grid service function calls or through SDEs. Grid services may publish SDEs, and other services can subscribe to these. Subscribers may be notified when the SDE's value changes.

The service model has many advantages over the job model that make it a more attractive candidate for grid-enabling VAIL. It is desirable to have nodes send status messages to the client as they are performing an analysis, and SDE notifications work well for sending such small messages. As discussed in the next section, SDEs are also useful when the client retrieves results. In addition, comparatively more documentation is available for the services model [10] than the job model in Globus 3.2.1. It also appeared that by adopting the Web Service Resource Framework [7] and Web Service Notification [8] specifications, the Globus team was

concentrating its efforts on the services component of Globus in the next major release, Globus Toolkit 4 [9]. Therefore, from a future supportability perspective, grid services seemed the prudent choice. Finally, the service model is more object-orientated and can be used to grid-enable existing Java applications more seamlessly than the job model.

As an aside, it is important to note that the job model has some advantages over the service model. It has been used more extensively, so more third party schedulers and tools are available for it. The authors anticipate from reading the Globus press releases that the next version of Globus will have a much more extensive and integrated set of tools, possibly including a scheduler framework. Finally, the job component of Globus has gone through many more stable releases and has been more thoroughly tested.

D. Grid Communication

Using grid services, Globus provides support for communication between different parts of the grid through three different mechanisms: function calls, SDE exposure, and SDE-based notifications. Each has advantages and disadvantages, and gVAIL utilizes all three.

1) Function Calls: Globus 3.2.1 provides support for function calls according to the standard grid service model: function (method) names for the grid service are exposed and can be invoked directly by the client. When the client makes a remote function call on the service, the service performs some computation and returns a value. The primary limitation of function calls is that the essential nature of most grid applications calls for a high degree of concurrency, while function calls block. This can be dealt with by using threading either on the client, the service, or possibly some combination of the two. gVAIL's client uses function calls to invoke the scheduler and accepts the blocking behavior, since it needs the scheduling results before it can proceed. It needs to communicate with multiple gVAILAnalyzer services in parallel, however, so gVAIL must use threading for this. For the sake of simplicity, gVAIL uses threading exclusively on the service end. gVAIL client makes a function call that returns immediately to start the analysis. Once the analysis is started on the grid nodes, the other two mechanisms are utilized for further communication.

2) SDE Exposure: Globus grid services also support SDEs, which behave essentially like public fields in a object. In this case, the service itself is the object. The service can store data in them, and the client can retrieve that data. Clients can also set the values of SDEs, although gVAIL does not utilize this feature. The gVAIL analyzer service stores the results it generates in SDEs. Once the client determines that a service has completed its analysis (see next section for how), it retrieves the results from the appropriate SDE on the service.

3) Notifications: In addition to simply storing and exposing data, SDEs form the basis for notifications in Globus 3.2.1. To receive notifications, the client subscribes to individual SDEs on a specific service. The service can, at any time, notify all listeners of a change to its SDEs, even if no change has actually taken place. These are *push* notifications in that the data stored in the SDE is sent to the client with the notification. *Pull* notifications differ in that the only data sent is a notice indicating that a change has occurred.

There are two points worth noting about notifications. First, they operate on the principle of *status updates*, meaning newer notifications will overwrite older notifications if they arrive at the client out of order. For example, if the service sends the notification messages AB, the client may receive either AB or BB. Thus, notifications are not a useful mechanism for passing discrete messages between the client and the service.

Second, the mechanism by which notifications are handled on the client is fairly complex and poorly documented, making it difficult to determine even what type of notification the client received, much less the value it stores. A method called `delieverNotification(...)` is called on the client and is passed an object of type `org.gridforum.ogsi.ExtensibilityType`, which encapsulates the SDE and its metadata. The `ExtensibilityType` class is largely undocumented, and extracting information from it is difficult at best. gVAIL uses notifications both to keep the client up-to-date on the status of the services and to inform the client when services complete.

E. Reliable Index Service

One of the goals of grid computing is to make computational resources as transparent and easy-to-use as the electrical power grid. In order for this to become a reality, however, the grid community needs to provide a reliable cataloging service for grid resources. Globus's index service falls short of this goal, and the authors were forced to use several work-arounds to make the cataloging of grid nodes more robust.

1) Dynamic Resubscription: One issue that complicated reliable indexing has to do with Globus's handling of SDE subscriptions. If an attempt at establishing a subscription fails, there is no convenient way to have Globus retry the subscription at regular intervals. This was a problem for gVAIL because it meant that the index service must start before the node information publisher service on any of the nodes. When node info publisher starts, it attempts to register the node with the index service by establishing a subscription between one of its SDEs, `gVAILNodeInfo`, and the index service. If the index service is not running, the subscription will fail, and Globus will never attempt it again. Thus, the index service will never know about the node. The authors solved this problem by using a timer in node info publisher to periodically renew the subscription, whether it is active or not. This solved another problem in that if the index service went down, all node information would be lost. Globus provides a solution to this through the use of the `xindiceEnabled` option to make the index service's database persistent between restarts. Using dynamic resubscription, however, this was unnecessary.

2) goodUntil Attribute: Essential for reliable dynamic indexing of grid resources is the ability to handle resource dropouts. The index service catalogs grid resources through

the use of SDE subscriptions, so it is essential that those subscriptions remain valid only for a certain amount of time. Nodes must periodically renew subscriptions, but if a node goes down and doesn't renew its subscription that data should disappear from the index service. However, Globus's mechanism for causing SDE subscriptions to expire leaves much to be desired. Each SDE subscription has a `lifetime` attribute that specifies how long the subscription should last. However, it seems that if the index service subscribes to the same SDE (`gVAILNodeInfo`) on multiple nodes [11], service data will disappear only after *all* `gVAILNodeInfo` subscriptions expire. If only one of the subscriptions expire, its data will still be published by the index service. Thus, if a node crashes and does not renew its `gVAILNodeInfo` subscription, the index service will still think it was available and will give this incorrect information to the scheduler.

To get around this, the authors uses the `goodUntil` attribute [12] associated with each SDE. When the node information publisher renews the index service's subscription to `gVAILNodeInfo`, it updates the SDE's `goodUntil` value to reflect the current time, plus a configurable lifetime for the data. When the scheduler receives a list of nodes from the index service, it checks to ensure that `goodUntil` for each node does not contain a time before the current time and discards any nodes with such `goodUntil` values.

IV. gVAIL Performance

A. Deployment Architecture

To test gVAIL's performance, it was deployed on four 2.8 GHz Pentium 4 machines with 640 MB RAM running Redhat Linux 9. Each machine ran Globus Toolkit 3.2.1 (web service base only) from a local drive. After this initial deployment, another sixteen machines (2.8 GHz Pentium 4's with 1 GB RAM) that form part of a Beowulf cluster at Harvey Mudd College were added. These machines used Redhat Enterprise Linux and the same versions of Globus mentioned above. Globus was placed on a network drive shared among all sixteen machines, and data files were distributed *a priori* on the shared drive. All twenty machines were connected via 100 Mbps ethernet switches to a 1 Gbps ethernet backbone.

B. Testing Apparatus

Software testing was performed using gVAILBatchModeClient, a special client designed to perform many consecutive gVAIL analyses. gVAILBatchModeClient reads an XML configuration file that describes the analyses to be run. It then executes each analysis sequentially.

A few factors that affect the runtime of gVAIL should be noted. First, VAIL uses short circuit evaluation to determine if a particular point can be identified as an anomaly, so the overall runtime is dependent on the number of anomalies in a file. Thus, if one node receives an unfair number of anomalies, it will take much longer than the rest of the nodes, delaying completion of the analysis. Finally, in order to minimize the systematic error that may be introduced by repeatedly using the same machine, the scheduler randomly selected which nodes to use.

C. Experimental Process

Four data files with varying total points and anomalous points were analyzed using a variable number of nodes. The characteristics of these files are shown in Table I. The number of nodes ranged from one to twenty in increments of one, and for each number of nodes, each file was analyzed twenty times. Finally, a serial VAIL analysis was performed on each file forty times to establish a performance baseline.

	Total points	Anomalous points
File 1	3,200,000	72
File 2	1,784,000	640
File 3	1,783,993	629
File 4	219,071	813

TABLE I
CHARACTERISTICS OF THE SAMPLE FILES

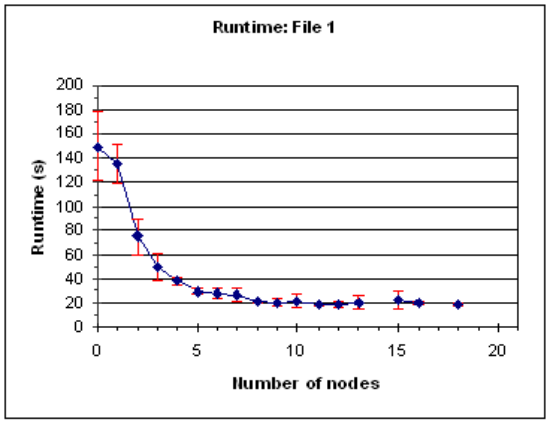
The runtimes (in seconds) for each file using 1 to 20 nodes are shown in Figure 5. On each graph, the serial VAIL performance baseline is shown as the "zero node" point. The speedup curves for all files are shown in Figure 6(a).

D. Results and Analysis

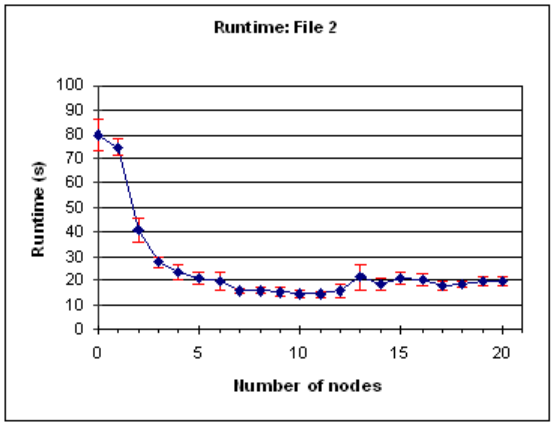
As expected, analyzing the smaller file using gVAIL on one node was slower than analyzing it using serial VAIL. However, analyzing any other file using gVAIL on one node was slightly *faster* on average than using serial VAIL. This is not statistically significant, however, because the baseline measurement's standard deviation in all cases nearly encompasses the standard deviation of the one node point.

Files 2 and 3 show similar results. This is expected because they contain comparable numbers of both total data points and anomalous points. However, note that File 2's absolute runtime is much lower due to the type and distribution of anomalies. For testing purposes, File 2 was generated artificially so that it would have an even distribution of anomalies. Thus, when the scheduler divides up the file based on number of points, it more evenly divides the actual work.

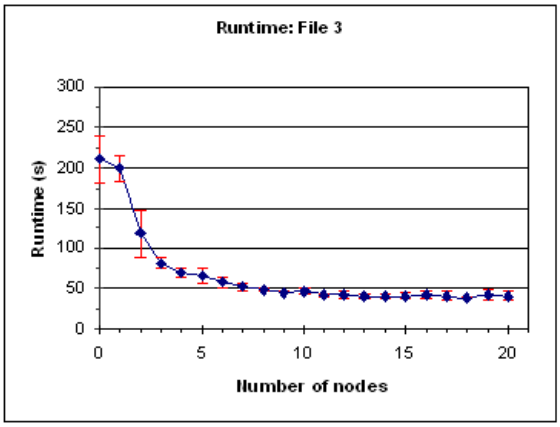
During the analyses of File 4, it started to behave erratically as the number of nodes increased beyond eight. This behavior was also seen, to a lesser degree, in the other waveforms. Part of this is due to the large number of back-to-back analyses performed. The erratic analyses were performed after 200 back-to-back analyses, and the performance of the grid degraded noticeably over time unless the nodes were restarted periodically. The period required varied with the size of the file. For instance, with the large file, services begin to fail at the file load stage after approximately eighty analyses. With the smaller file, services began to fail at this stage after



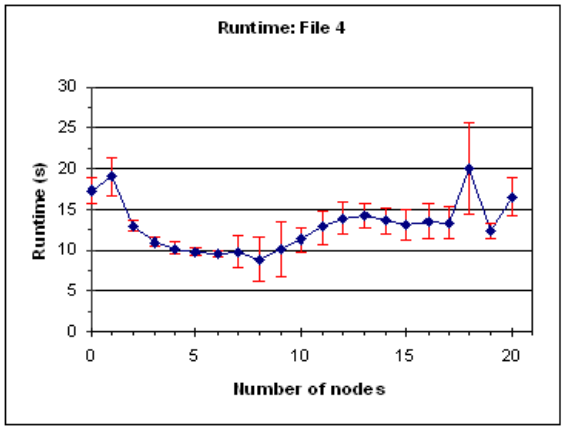
(a)



(b)

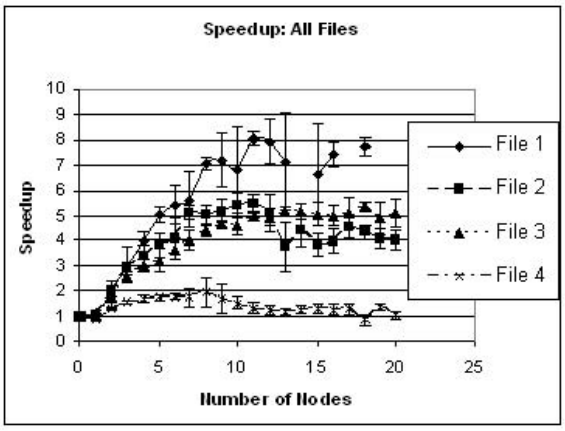


(c)

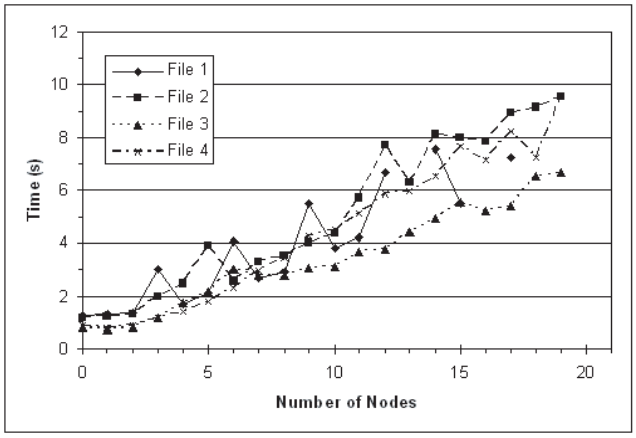


(d)

Fig. 5. Runtime Results



(a)



(b)

Fig. 6. Speedup and Communication Time Graphs

approximately five hundred analyses. The dependence on file size suggests that garbage collection is not working perfectly. It is also likely that some of the erratic behavior was caused by load on the Beowulf Cluster nodes from other projects. Load on the Beowulf nodes would produce more pronounced aberrations as the number of nodes utilized increased.

However, most of the contribution to increasing runtime seems to be communication costs, as demonstrated by Figure 6(b). Also, from Figure 6(b) it is possible to observe that the communications time is file size independent. Thus, a more advanced scheduler for future work in this project might consider how to balance communication cost with analysis

cost for more efficiency.

Overall, grid-enabling VAIL increased its performance. Moreover, the number of nodes at which speedup reaches a plateau varies with file size and anomaly distribution, as expected.

V. Next Steps

As indicated in the Introduction, gVAIL is the first part of an effort to create a service architecture for VISPERS using the Globus toolkit. Many of the VISPERS tools will benefit from parallelization. More importantly, there are a large number of telemetry streams for each launch that could be split among several grid nodes. The goal here is that from the human analysts perspective, they select a sensor and process the data using VISPERS to clean-up the data (VAIL and other tools), then they filter and run any appropriate analysis tools. They can see that the tools run more quickly, but the way the speedup is achieved is invisible to them. This work will be reported on in a future paper.

The next step in this task is to create a grid workflow engine to manage both the VISPERS applications and to ensure that the individual tools use the grid effectively. Since the gVAIL project was started, there have been significant enhancements to VAIL and other tools that will help with the data clean-up have been developed or enhanced. The goal will be to make the entire data clean-up process run seamlessly from data acquisition using dbFront, through anomaly removal and the consolidation of multiple telemetry streams into one *good* stream. Once that stream is created, individual analysis tools can be invoked, the results analyzed, and, if necessary, the data can be re-scrubbed based on those results.

This project was undertaken after the release of Globus 3.2.1 but before the release of Globus 4.0. In May 2004, the Web Service Resource Framework (WSRF) and Web Service Notification (WSN) specifications were submitted to OASIS [13] for comments, and the Globus team decided to develop a refactored version of the Globus services component based on that standard. Many of the trials and tribulations that were endured during this project can hopefully be avoided by utilizing Globus 4.0. According to the Globus website, Globus 4.0 will be much easier to install and use, will include documentation, and will be standards-compliant. Because of the switch to web services in Globus 4.0, one will be able to develop grid applications in any language using familiar tools. Therefore, the next step in this project is to port gVAIL to Globus 4.0.

VI. Conclusion

Grid-enabling VAIL using Globus Toolkit is a viable way to improve performance. Using as few as eleven grid nodes, gVAIL's runtime improved by a factor of eight. With little prior grid computing experience, however, the authors found the grid-enablement process to be difficult using Globus 3.2.1. This was due in part to its very steep learning curve, sparse documentation, unintuitive handling of SDE subscription lifetimes, and awkward notification mechanisms. Globus shows much promise, however, and the authors believe that the Globus developers will address these issues in future versions.

In the larger context, however, our experience shows that the service-oriented architecture approach, using grid computing tools, can provide a more flexible system design, in addition to improved performance and increased utilization of resources. With proper communication and execution scheduling, performance and utilization could be improved even further. We also wish to emphasize that gVAIL is only one component in the larger VISPERS tool suite that we are currently bringing under the control of a workflow manager where we will investigate the use of integrated communication and execution scheduling. When this is done, we expect to have a powerful yet flexible system that can process more data more quickly than previously possible, enabling a greater understanding of the launch vehicle environment.

References

[1] Bradford, K. B., Wong, D. and Bartosisk, J., July 2002, "The Vibroacoustic Intelligent System for Predicting Environments, Reliability and Specifications (VISPERS)," Proceedings of the Ninth International Congress on Sound and Vibration.

[2] Globus Toolkit 3.2.1 Documentation. http://www-unix.globus.org/toolkit/docs/3.2/

[3] Foster, Ian and Kesselman, Carl, ed., *The Grid 2*, Morgan Kaufmann, United States, 2004.

[4] WS Information Services: Key Concepts. http://www-unix.globus.org/toolkit/docs/3.2/infosvcs/ws/key/index.html

[5] GRAM: Key Concepts. http://www.globus.org/toolkit/docs/3.2/gram/key/index.html

[6] *The Open Grid Services Architecture, Version 1.0*. Global Grid Form Document GFD-I.030. 29 January 2005. http://www.ggf.org/documents/GWD-I-E/GFD-I.030.pdf

[7] OASIS Web Services Resource Framework (WSRF) TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[8] OASIS Web Services Notification (WSN) TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

[9] GT4.0 Common Runtime Components. http://www-unix.globus.org/toolkit/docs/4.0/common/key/index.html

[10] Globus Toolkit 3 Programmer's Tutorial. http://gdp.globus.org/gt3-tutorial/

[11] *Globus-discuss* Mailing List Archive. http://www-unix.globus.org/mail_archive/discuss/2004/04/msg00335.html

[12] Index Grid Services Using Globus Toolkit 3.0. http://www-106.ibm.com/developerworks/grid/library/gr-indexgrid/

[13] OASIS. http://www.oasis-open.org/home/index.php