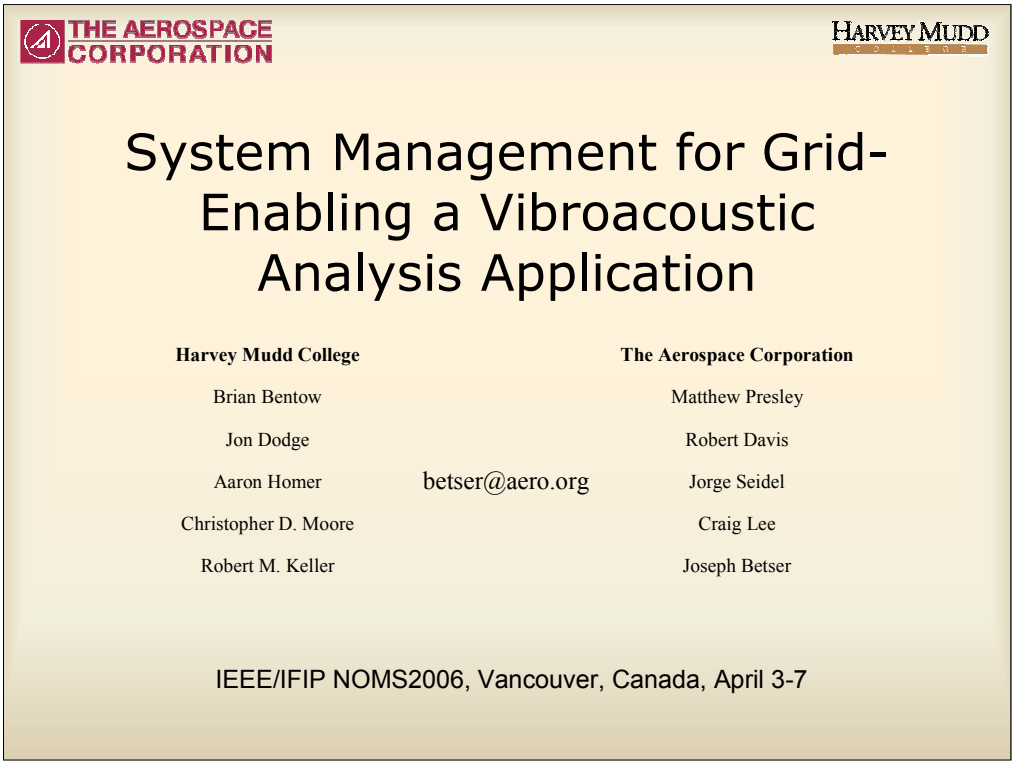


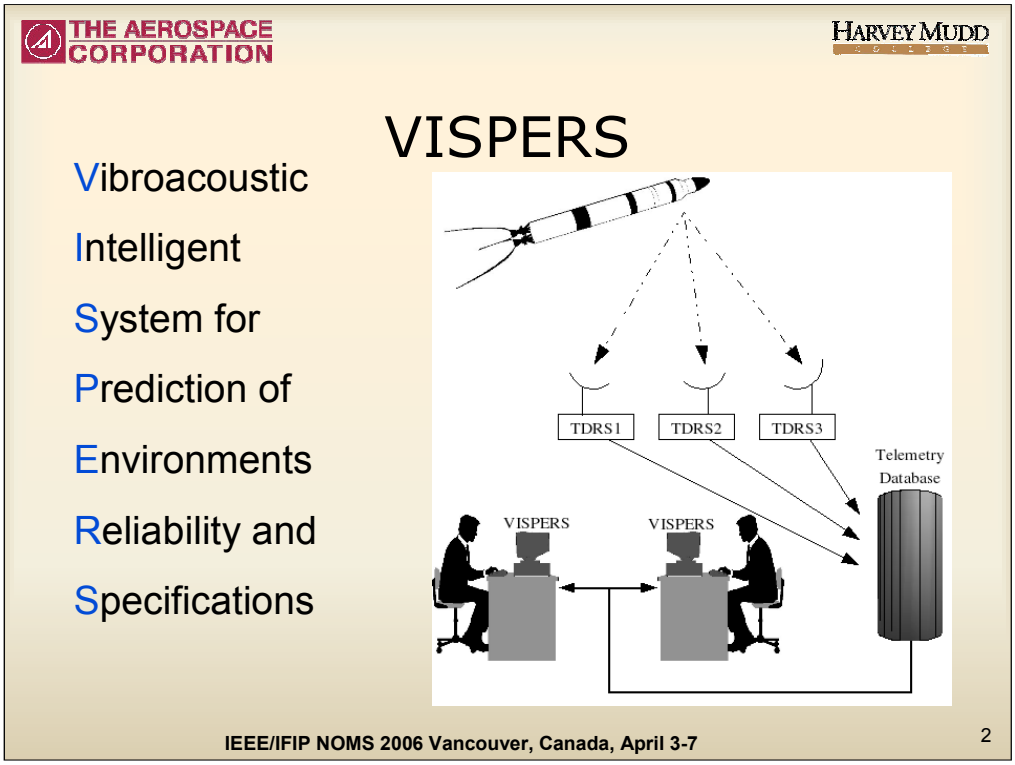
Keywords

System Management, Grid Computing, Service Oriented Architecture, Telemetry, Performance, Workflow Management

Abstract

System management aspects are described for 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 enterprise 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.

1-4244-0143-7/06/$20.00 ©2006 IEEE

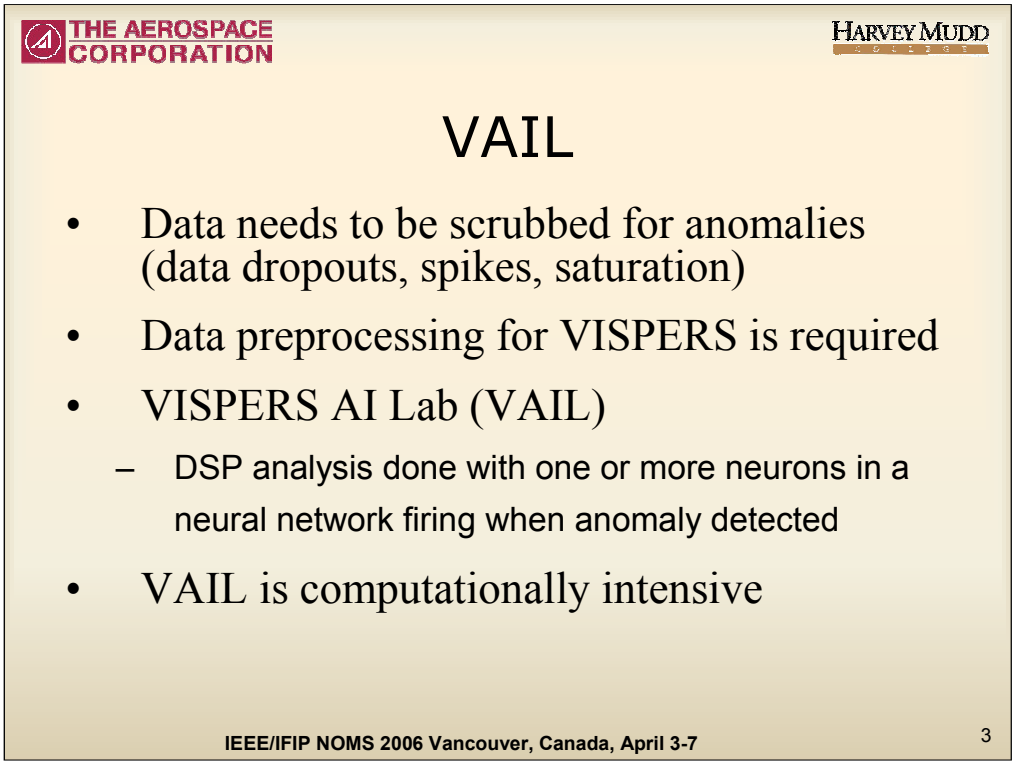


VISPERS

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. Along these lines, the Aerospace Corporation is developing a tool called VISPERS. The purpose of VISPERS is to analyze data recorded by sensors during a launch in order to better understand the vibroacoustic environment. The end goal is to improve vehicle and payload survival [1].

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 the figure. 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 consolidated into a single, best data stream prior to final processing. It is possible that hundreds of telemetry streams may have to be consolidated and analyzed to identify episodes of excessive vibration during launch.

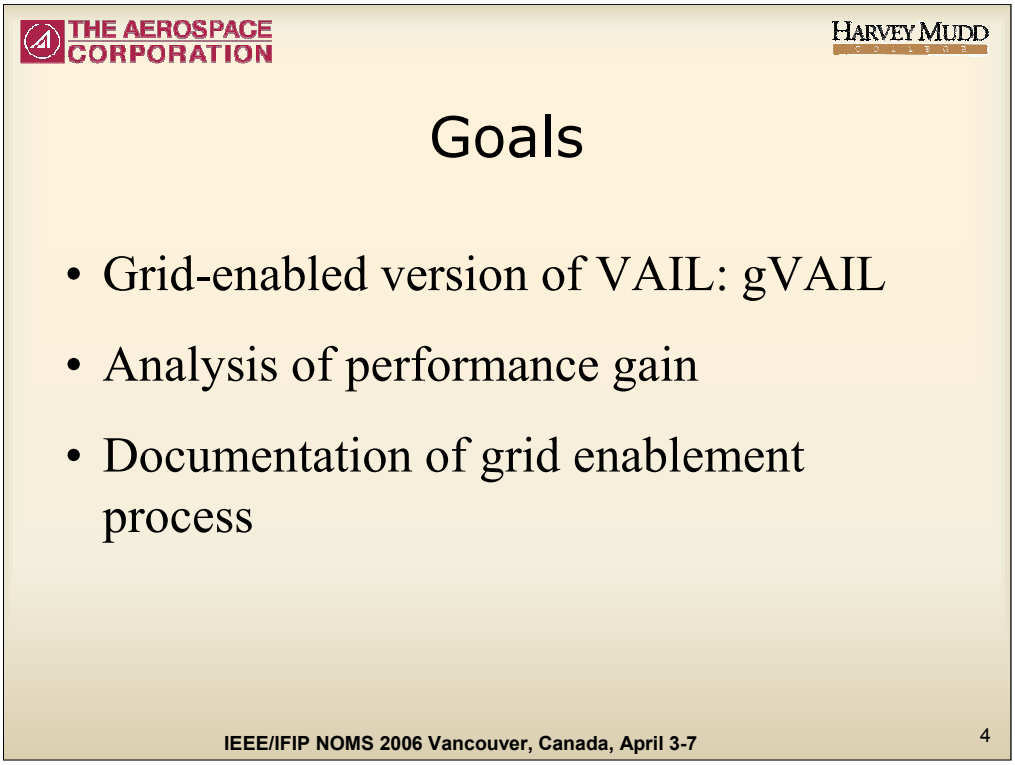
1-4244-0143-7/06/$20.00 ©2006 IEEE



VAIL

Telemetry streams include static and other anomalies and thus must be scrubbed for noise before final processing. For example, the signal may briefly disappear (drop out) due to the antenna on the spacecraft rotating away from the receiving station. The VISPERS component that performs such preprocessing is called VAIL (VISPERS AI Lab). VAIL does a DSP analysis on the noisy data using a neural network in which one or more neurons fire when an anomaly is detected. This, however, is computationally intensive and can benefit from grid-enablement.

1-4244-0143-7/06/$20.00 ©2006 IEEE



Goals

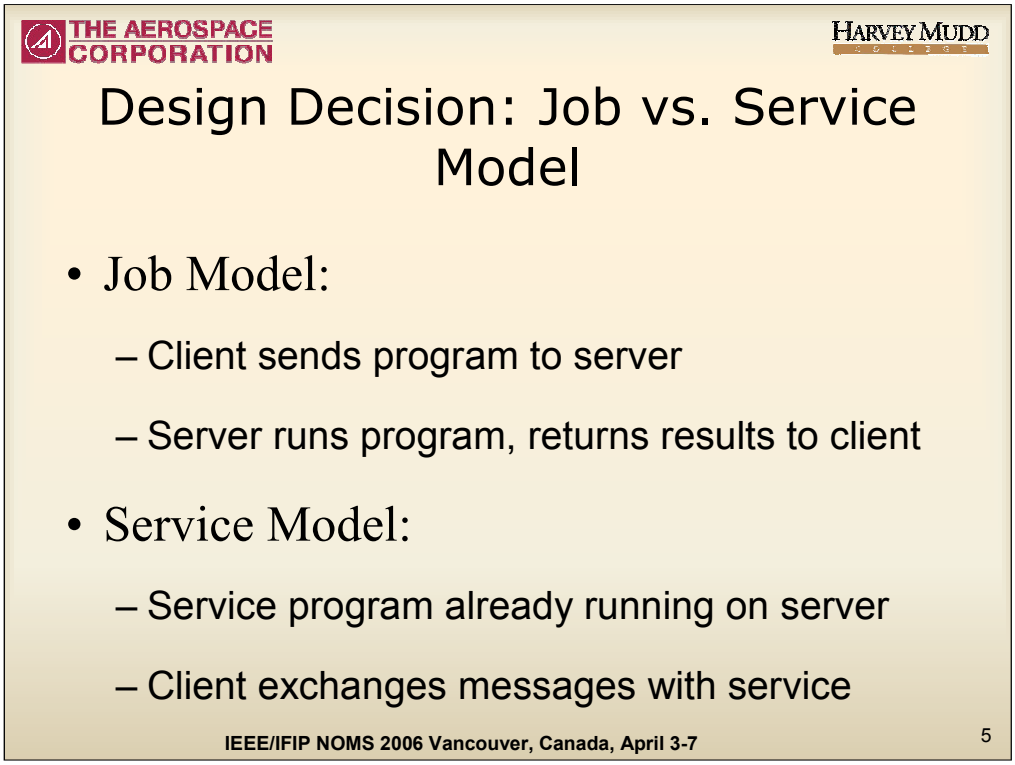
Our goals for the project were as follows:

-Grid-enable VAIL

-Perform a case study on grid technologies, focusing specifically on Globus Toolkit 3.2.1 [2]

-Document issues encountered and lessons learned in the grid-enablement process so that future efforts may benefit from our experience

1-4244-0143-7/06/$20.00 ©2006 IEEE



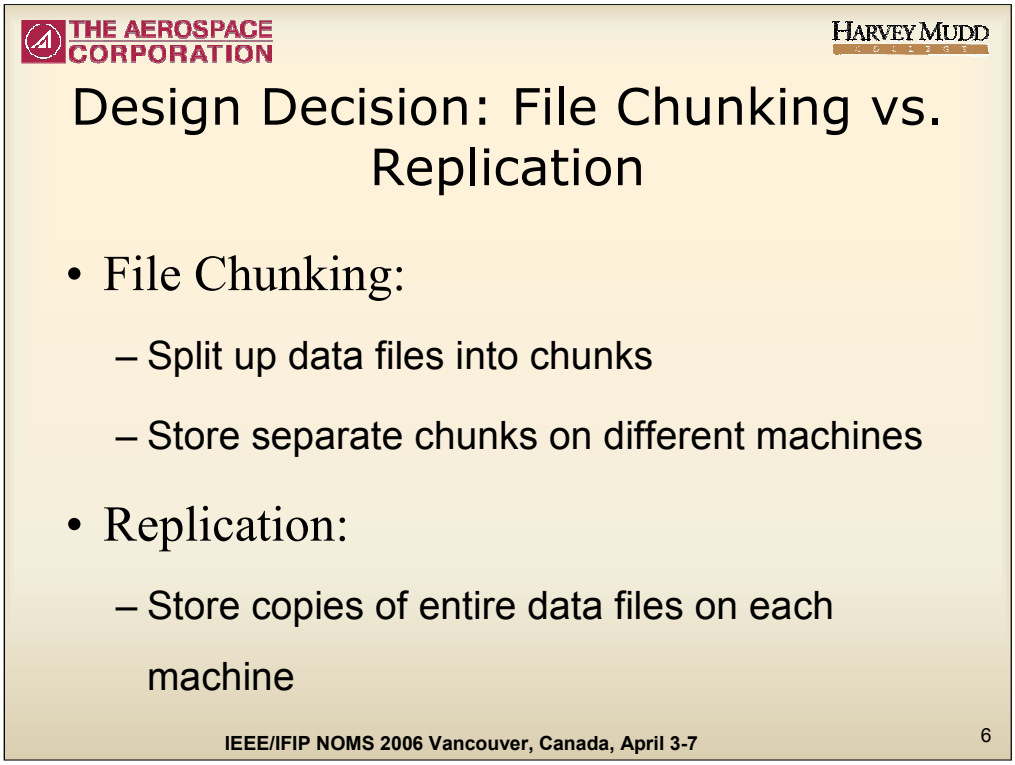
Design Decisions

We made a number of design decisions during the course of the project. The major ones are discussed here and one of the following slides.

Job vs. Service Model

We investigated the use of two different distributed computing models: the job model [5] and the service model [6]. In the job model, a client sends the executable program it wishes run to the server, which executes the program and sends the results back to the client as a text stream. The client must then parse this stream to extract the desired information. In the service model, the program to be executed is pre-installed on the server and runs as a service, awaiting requests. The client exchanges messages with this service in a pre-defined format. The grid community seems to be moving toward the service model, so we chose this model for future supportability reasons [9]. ***It should also be noted that the service model relies on service discovery and registration so that functionality is better abstracted and more independent of functionality than with the job model.***

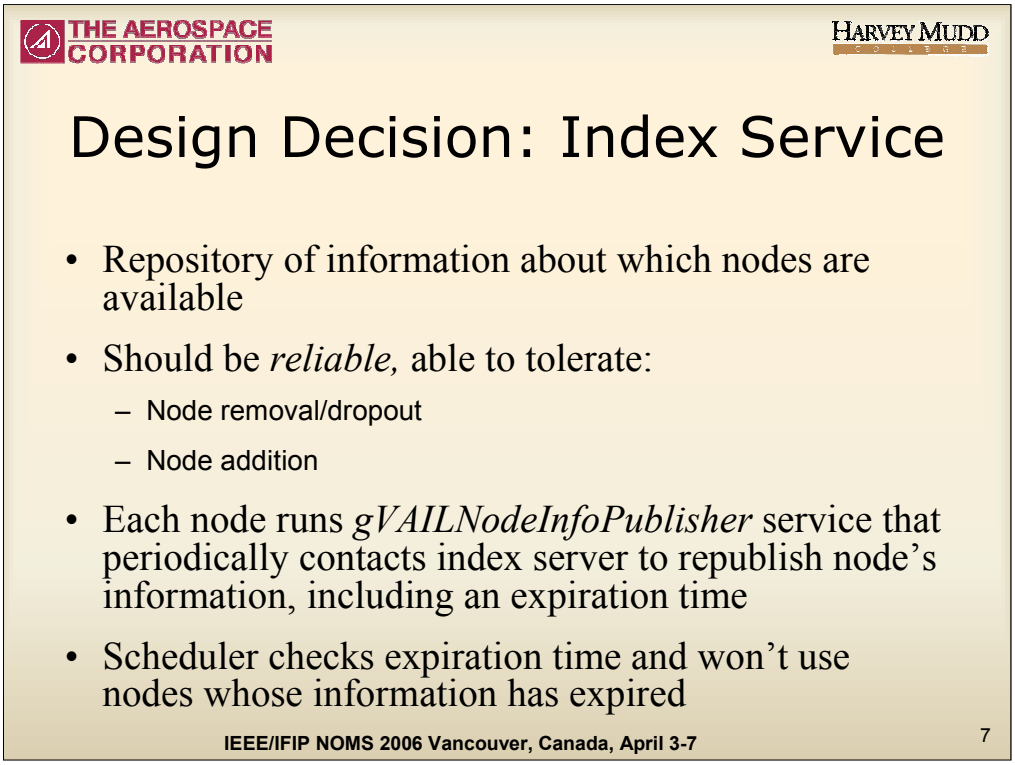
1-4244-0143-7/06/$20.00 ©2006 IEEE



File Chunking vs. Replication

Another design decision we had to make was whether to split each data file across multiple machines (the "file chunking" approach) or to store a complete copy of each file on each machine (the "replication" approach). We chose the replication approach for several reasons. First, file chunking severely complicates scheduling as the scheduler must know which parts of each file are on each node. Secondly, chunking complicates file distribution as files must be split into the appropriate number of chunks before being copied to each node. Thirdly, chunking complicates the process of removing nodes from the network as appropriate steps must be taken to redistribute a node's chunks when it is removed. Finally, disk space is relatively inexpensive, so the overhead of storing entire copies of each data file on each machine is tolerable. It should be noted that for replication to work, the frequency with which files are replicated across nodes must be much greater than the frequency with which the data changes. This was not a problem for this project.

1-4244-0143-7/06/$20.00 ©2006 IEEE

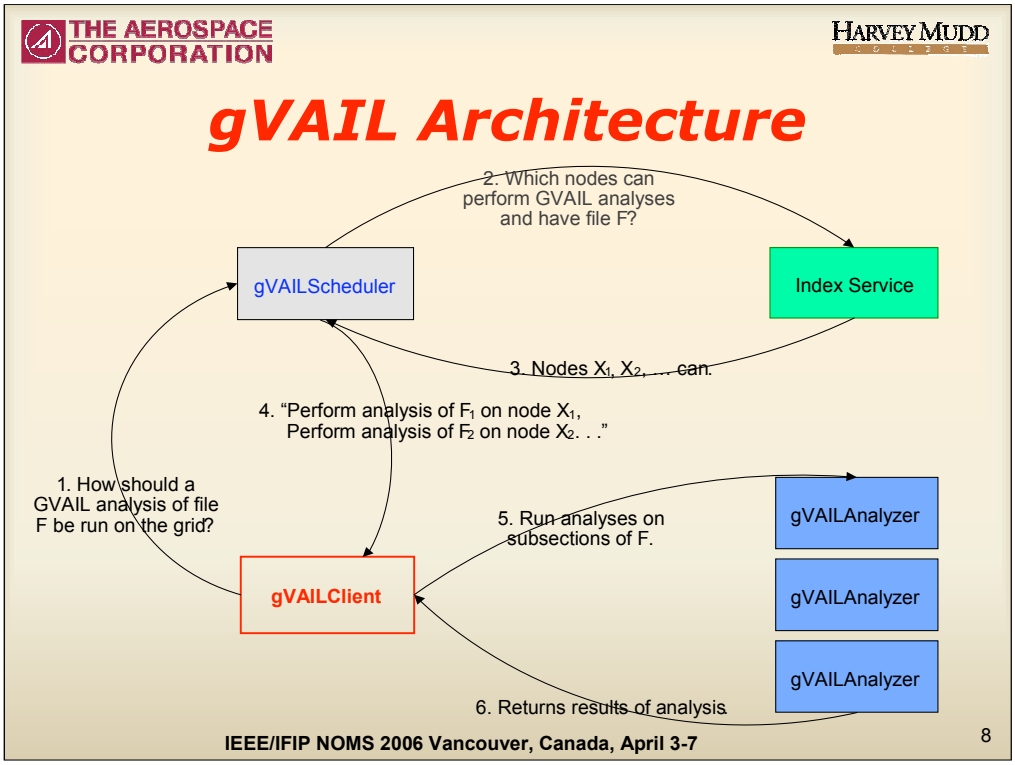


Node Indexing

We wanted gVAIL to be able to function on a dynamic grid topology, so it was essential that node indexing be reliable. That is, when a node is removed or crashes, it should no longer be used to schedule analyses, and when a node is added, it should be used to schedule future analyses. To ensure that it is indexed reliably, each node runs a service called gVAILNodeInfoPublisher which periodically contacts the index server (a machine running Globus's Index Service [4]) to republish the node's information. Thus, when a node is added, its gVAILNodeInfoPublisher service will automatically notify the index server of its existence. Among the information that gVAILNodeInfoPublisher exposes is an expiration time for its data. When the scheduler splits up a task, it checks each node's data to make it is still valid. If it is not, the scheduler doesn't use the node. Thus, if a node is removed or crashes, the expiration time of its information will pass and the scheduler will cease to use it when dividing analyses [12].

This TTL mechanism does have a drawback. If a crashed node's information does not expire before the next analysis is scheduled, the the scheduler will instruct the client to use that node. The client's request to the node will fail, but the user will be notified. The user can then resubmit the analysis request to the scheduler, by which time the crashed node's information may have expired. We felt that this drawback was acceptable, however, given that the alternative is to have the scheduler ping each node to verify that it is alive before dividing the analysis. This would introduce significant communication overhead for grids with a large number of nodes.

1-4244-0143-7/06/$20.00 ©2006 IEEE

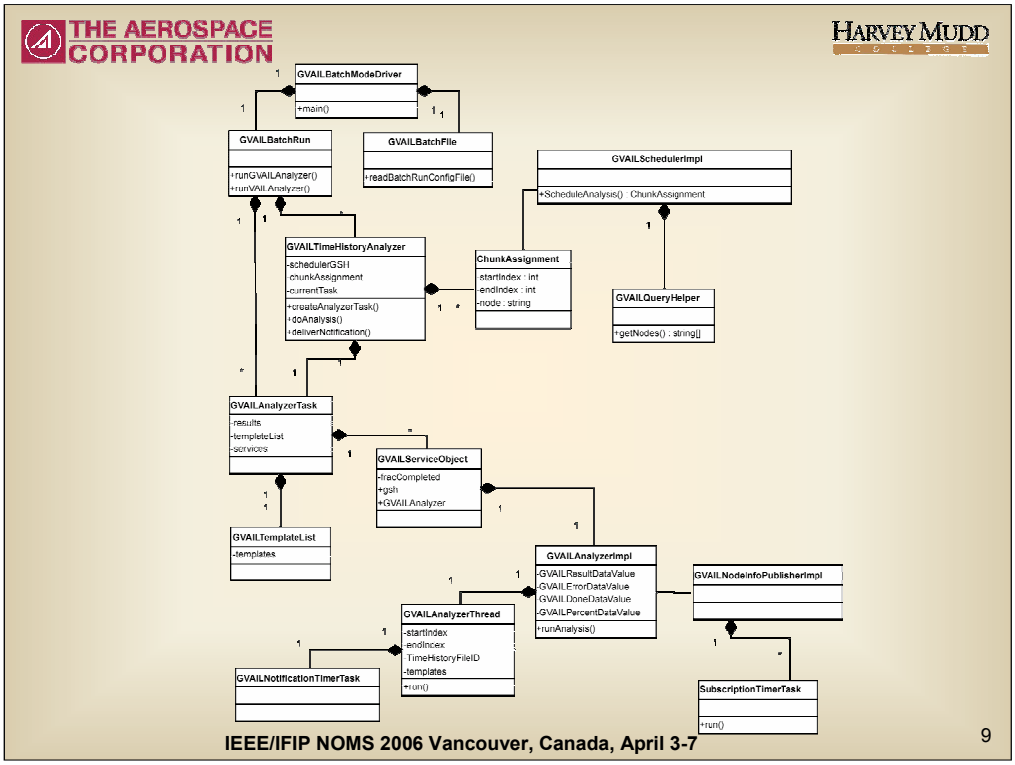


Final Design

The following sequence of steps is taken then the user tells the client to begin an analysis:

1. The client asks the scheduler how it should split up the analysis across the grid.
2. The scheduler talks to the index service to determine which nodes are available to perform the analysis.
3. The index service responds with a list of nodes.
4. The scheduler decides which node should analyze which part of the file and returns this information to the client.
5. The client then contacts each node and instructs it to begin analyzing the correct section of the file.
6. Nodes return results to the client.

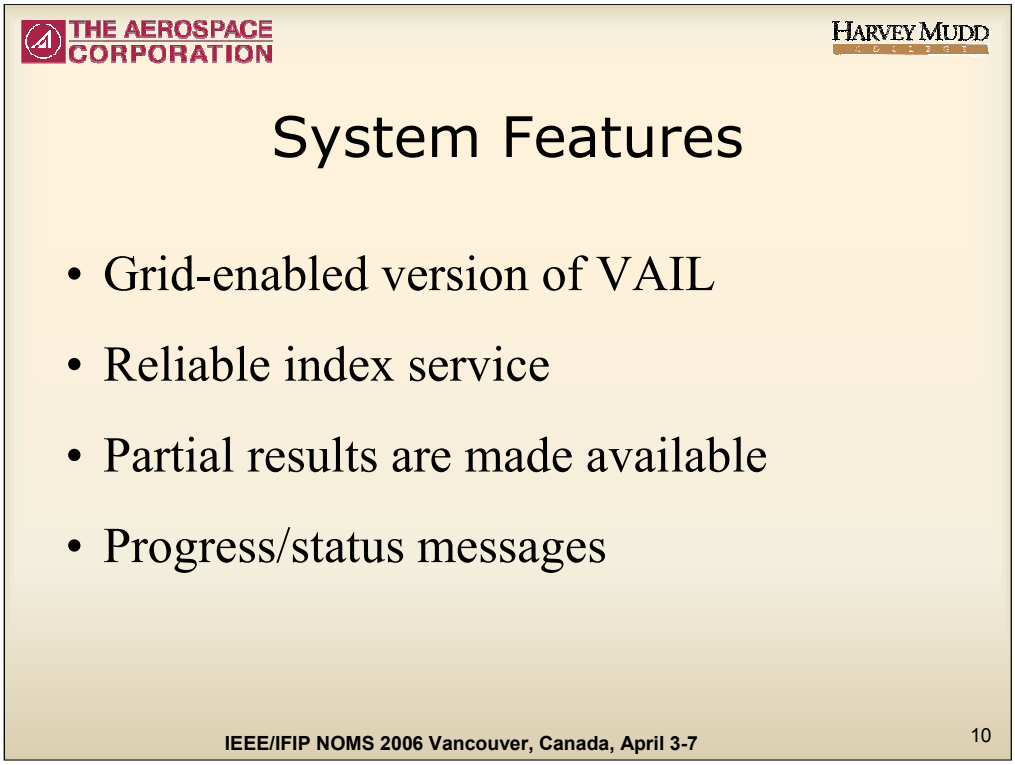
1-4244-0143-7/06/$20.00 ©2006 IEEE



UML Class Diagram

The gVAIL class structure is depicted in the figure. Classes GVAILBatchModeDriver, GVAILBatchRun, and GVAILBatchFile compose gVAILBatchModeClient, a special client built for testing purposes and 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. 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 node information publisher is implemented in class GVAILNodeInfoPublisherImpl and uses a timer (implemented in SubscriptionTimerTask) to periodically resubscribe itself to the index service.

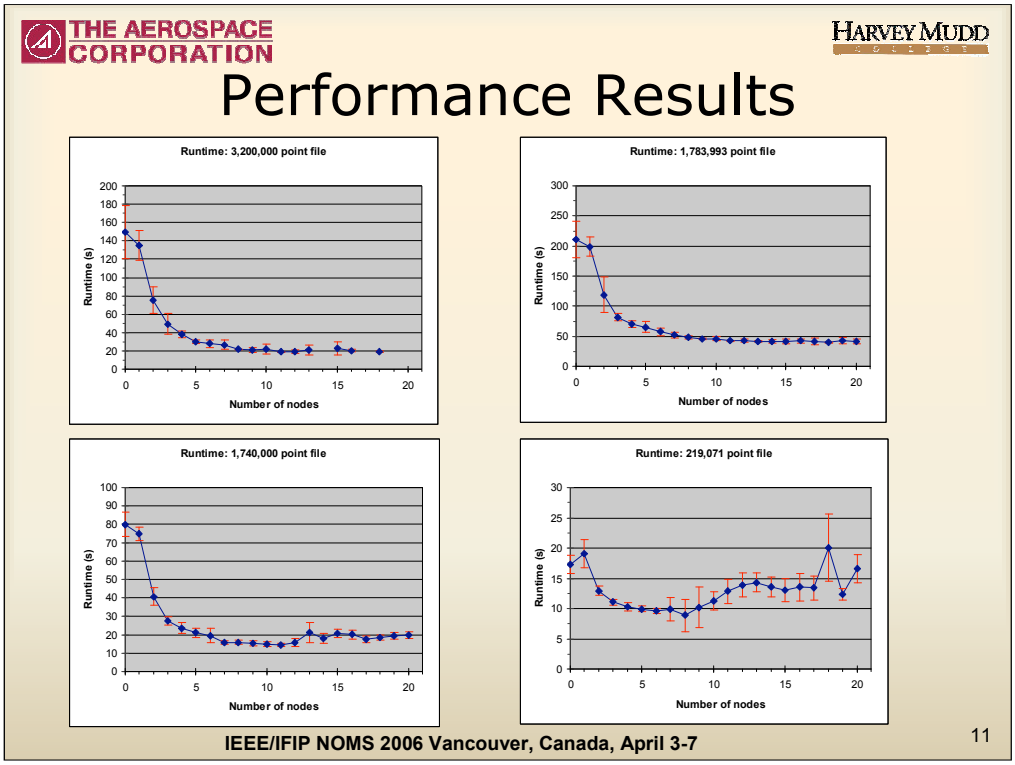
1-4244-0143-7/06/$20.00 ©2006 IEEE



System Features

gVAIL features a grid-enabled version of the VAIL code, along with the reliable indexing mechanism discussed previously. In addition, the gVAILAnalyzer service on each node publishes its current results as service data, and clients are free to query this data at any time. In addition, each node exposes service data that indicates the percentage of its analysis it has completed and updates this periodically. At the beginning of an analysis, the client registers as a listener of this data, and whenever the data is updated, the client receives the new value. The client uses this notification mechanism not only to keep the user informed as to the progress of the task but also to ensure that each node is still alive. If a node doesn't send a status update for a specified period of time, the client assumes the node has died and alerts the user.

1-4244-0143-7/06/$20.00 ©2006 IEEE



Performance Testing Experiments

-Run an analysis on a given file and number of nodes 15-20 times

-Subset of grid nodes are selected randomly

-Computed statistical summaries

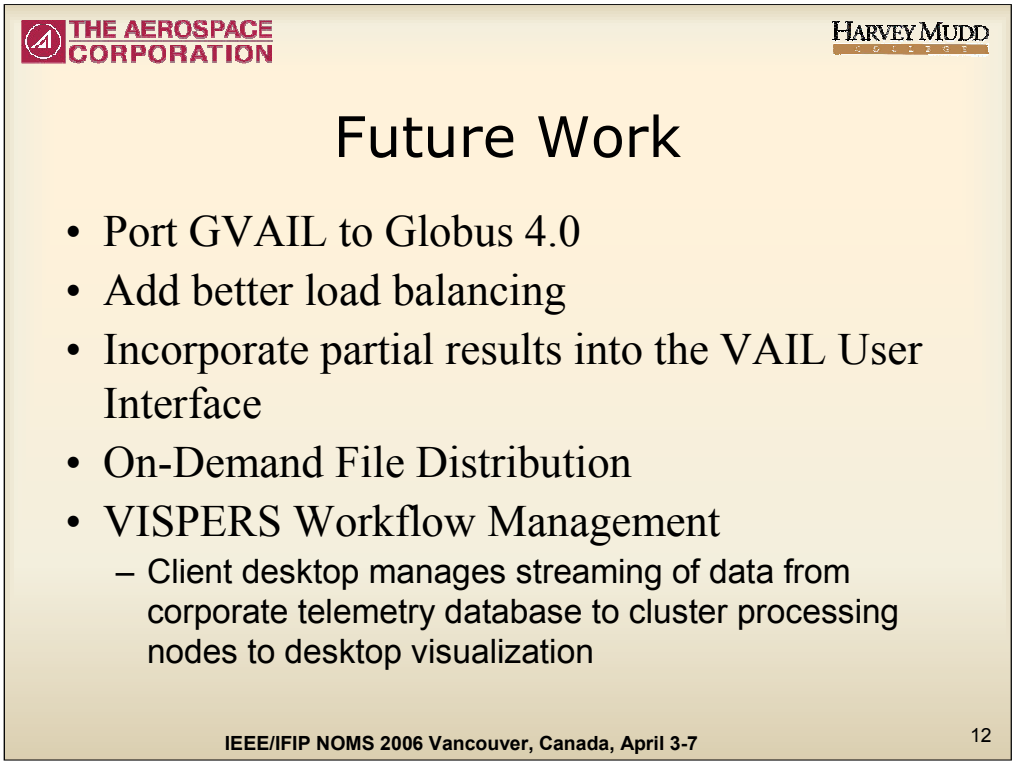
Performance Results

The above graphs show gVAIL's runtime for different files, plotted as a function of number of nodes. On each graph, the zero node point gives standard VAIL's runtime when analyzing the file, used to establish a baseline. Of note is that on a number of the files, serial VAIL's runtime was higher than gVAIL's using one node. One would expect the opposite to be true due to the communication overhead incurred using gVAIL. However, we don't feel that this is statistically significant because the standard deviation of the baseline measurement nearly encompasses the standard deviation of the single node gVAIL measurement in most cases.

The erratic behavior seen in the bottom right graph (smallest file) is likely due to communication time and load on the network. We found that communication time was independent of file size so that it formed a larger fraction of the total analysis time for smaller files. Thus, any erratic network load would have a much larger impact on small file performance.

Note that although the top right and bottom left graphs show runtimes for similarly sized files, the runtimes are noticeably different. The bottom left file was artificially generated so that the anomalous points were evenly distributed. Our scheduling algorithm divides a task such that each node analyzes the same number of points, but VAIL's neural network code runs slower on segments with a large number of anomalies. Thus, our scheduler more evenly divides the actual work on files with evenly distributed anomalies, leading to better performance when analyzing such files.

1-4244-0143-7/06/$20.00 ©2006 IEEE



Future Work

In the future, gVAIL could be improved by:

-Porting it to Globus 4.0 [9], which should be:

-Easier to install

-Better documented

-Standards compliant with web services

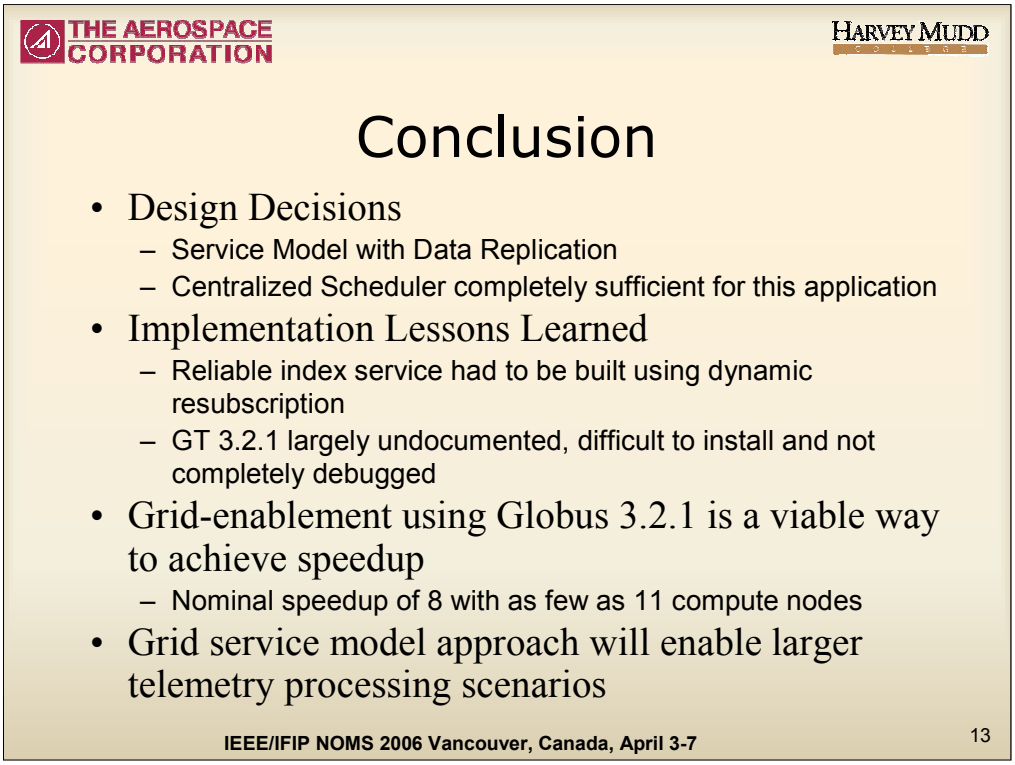
-Improving our scheduling algorithm so that it takes into account information that may be known about anomaly density in different regions of a data file

-Incorporating the retrieval and display of partial results into the VAIL user interface

-The ability to distribute files on-demand

-Adding workflow management capability to the client

1-4244-0143-7/06/$20.00 ©2006 IEEE



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://wwwunix.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://wwwunix.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.oasisopen.org/committees/tc home.php?wg abbrev=wsn

[9] GT4.0 Common Runtime Components. http://wwwunix.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://wwwunix.globus.org/mail archive/discuss/2004/04/msg00335.html

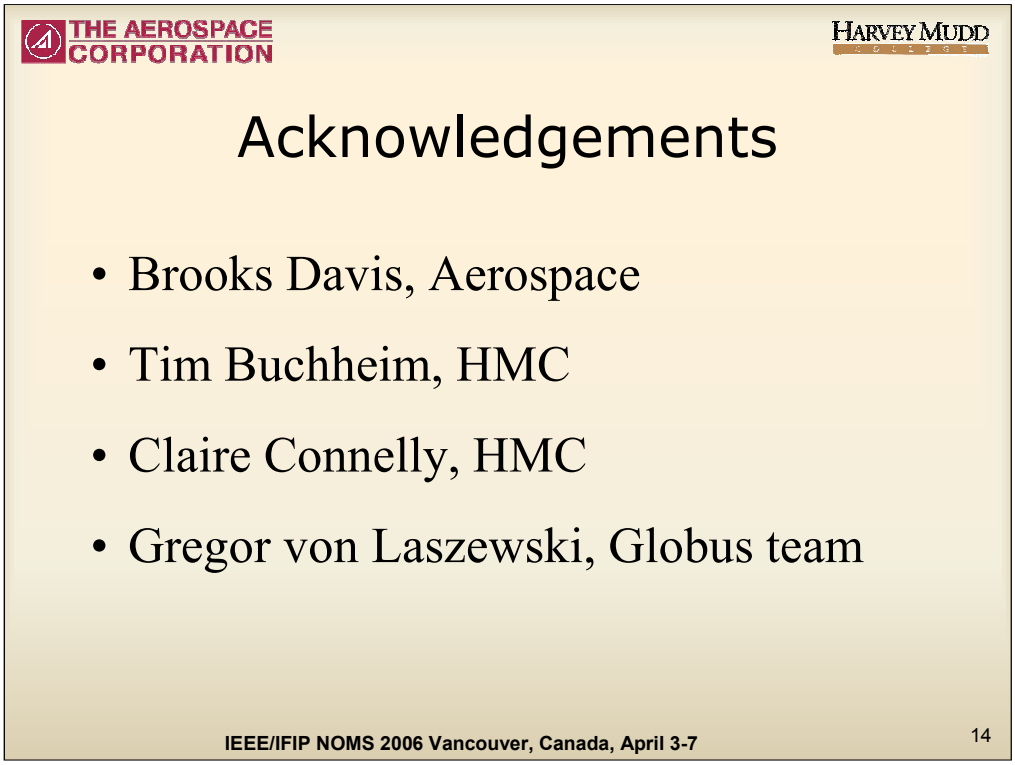
[12] Index Grid Services Using Globus Toolkit 3.0. http://www-106.ibm.com/developerworks/grid/library/gr-indexgrid/

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

[14] Bentow, B., Dodge, J., Homer, A., Moore, C., Keller, R.M., Presley, M., Davis, R., Seidel, J., Lee, C., Betser, J., November 2005, "Grid-Enabling a Vibroacoustic Analysis Application", Proceedings of the Sixth IEEE/ACM International Workshop on Grid Computing

[15] Bentow, B., Dodge, J., Homer, A., Moore, C., Keller, R.M., Presley, M., Davis, R., Seidel, J., Lee, C., Betser, J., "Lessons Learned from Grid-Enabling a Vibroacoustic Analysis Application", Invited paper, To appear in the International Journal of High Performance Computing and Networking, special issue Fall 2006

1-4244-0143-7/06/$20.00 ©2006 IEEE



Acknowledgements

We'd like to thank the following people:

-Brooks Davis, also from The Aerospace Corporation

-Tim Buchheim and Claire Connelly, the CS and Math system administrators at Harvey Mudd College

-Gregor von Laszewski of the Globus team