GA–A23834

A NEW SYSTEM TO MONITOR DATA ANALYSES AND RESULTS OF PHYSICS DATA VALIDATION BETWEEN PULSES AT DIII–D

by
S. FLANAGAN, J.M. SCHACHTER, and D.P. SCHISSEL

FEBRUARY 2003

DISCLAIMER

GA–A23834

A NEW SYSTEM TO MONITOR DATA ANALYSES AND RESULTS OF PHYSICS DATA VALIDATION BETWEEN PULSES AT DIII–D

by
S. FLANAGAN, J.M. SCHACHTER, and D.P. SCHISSEL

This is a preprint of a paper to be submitted for publication in *Phys. Rev. Lett.*

Work supported by
the U.S. Department of Energy under
DE-AC03-99ER54463 and DE-FC02-04ER54698

GENERAL ATOMICS PROJECTS 30033 AND 30200
FEBRUARY 2003

ABSTRACT

A Data Analysis Monitoring (DAM) system has been developed to monitor between pulse physics analysis at the DIII–D National Fusion Facility (http://nssrv1.gat.com:8000/dam). The system allows for rapid detection of discrepancies in diagnostic measurements or the results from physics analysis codes. This enables problems to be detected and possibly fixed between pulses as opposed to after the experimental run has concluded thus increasing the efficiency of experimental time. An example of a consistency check is comparing the experimentally measured neutron rate and the expected neutron emission, RDD0D. A significant difference between these two values could indicate a problem with one or more diagnostics, or the presence of unanticipated phenomena in the plasma. This new system also tracks the progress of MDSplus dispatched data analysis software and the loading of analyzed data into MDSplus. DAM uses a Java Servlet to receive messages, CLIPS to implement expert system logic, and displays its results to multiple web clients via HTML. If an error is detected by DAM, users can view more detailed information so that steps can be taken to eliminate the error for the next pulse.

I. INTRODUCTION

For the past several years at DIII–D, an increasing amount of computational data analysis has been taking place between pulses. With an increasing amount of data available, more centralized monitoring capabilities are required to maintain a manageable workload by the physics and operation teams. At the DIII–D National Fusion Facility, the Data Analysis Monitoring (DAM) system has been designed in order to monitor between pulse physics analysis. This new system was developed with the capability to detect discrepancies in the results of physics analysis codes, to track data loading, and to evaluate the acquisition time of data.

DAM is designed to be a portable system that can be installed and run on any computer with modest hardware requirements. To obtain this level of portability, the core of DAM has been developed as a servlet using the Java programming language. The servlet itself uses the Java Expert System Shell (Jess)[1] which provides DAM with the capacity to "reason" using knowledge supplied by facts and rules. When a new fact is declared, the fact is evaluated against DAM's rules, using the famous Rete algorithm.[2]

Information is received by DAM in the form of a message. These messages can be sent from any code on any machine, however, DAM will only acknowledge this information if it is received from a valid host. If the machine is not recognized as a valid host, this message will simply be ignored. This allows any and all off site collaborators to provide DAM with information, simply by having their host added to the list of valid information sources. In addition, this provides DAM with the ability to accept facts and evaluate rules without sacrificing security or creating platform and application dependencies.

The user interface, which shows the pulses and their corresponding facts for the current day, is provided through Hypertext Markup Language (HTML) generated by the Java Servlet. This interface enables multiple users to connect to DAM without regard to their location. In other words, there is no special client software needed to use the system, and any user may obtain information for the current DIII–D experiment. A user simply needs to launch a web browser and connect to DAM via the host name and port number of the DAM server. When DAM receives new information, the system will automatically refresh each client, via server push, and display the current information in real-time. This allows all users, both on site and off site collaborators, to quickly and conveniently monitor the analysis process at DIII–D in real-time (http://nssrvl.gat.com:8000/dam).

The actual display for the monitoring system provides several functions. Besides alerting users to errors and discrepancies, DAM allows users to view the logfiles that were created by the specific code that reported to DAM. This too is done through the web interface. In order to exclude certain files from becoming public, each log file must contain a standardized header that identifies it as a file that is allowed to be viewed through DAM. System administrators running DAM would, of course, be encouraged to evaluate file permissions on any critical files.

The Data Analysis Monitoring system should provide several benefits to any site amongst the fusion community. This new system can be used to evaluate data loading, and acquisition timing. It can be used to discover discrepancies or errors in data acquisition and physics analysis. It can even provide an incentive to improve physics codes and can assist users in pinpointing data analysis issues, both good and bad.

II. THE DATA ANALYSIS MONITORING (DAM) SYSTEM

Typically, any expert system contains a knowledge base, and a set of rules to which its knowledge is applied. More precisely, the knowledge base of a program is composed of a set of facts. Rules then, take action based on one or more of these facts. DAM provides this capability by utilizing the Java Expert System Shell, Jess.

Jess is a rule-based expert system shell, which uses the well known Rete algorithm. Rules contain a left hand side (LHS), which is checked against the knowledge base, and a right hand side (RHS), which is executed should the rule be satisfied.[1] In the Rete algorithm, only new facts are tested against the LHS of a rule to which they are most likely to be relevant with.[1,2] This means that the algorithm does not waste resources checking facts it already knows, and won't waste time repeating tests across iterations. The algorithm is implemented by building a network of nodes (Fig. 1), each representing one or more tests found in a rule LHS. Facts being added to the knowledge base are processed based on this network of nodes. When a fact, or set of facts, filters all the way down to the bottom, the associated rule will have its RHS fired, or executed. With the addition of Jess, this approach provides DAM with "reasoning" capabilities.

To provide a concrete example of how reasoning functions are accomplished, let us use an every day occurrence. Figure 1 represents the thought process of a person when sitting at a traffic light. When the left hand turn light becomes green (X), then it should be safe for a car to turn. However, before that car may turn, oncoming traffic must also be clear (Y). When both X and Y have occurred, then the driver would check to make sure that there are no pedestrians walking across the street (Z). Then, when all knowledge has been taken into account and the LHS [(X+Y)+Z], has occurred, then the RHS may be fired. In this case, the car would be able to make a left turn.

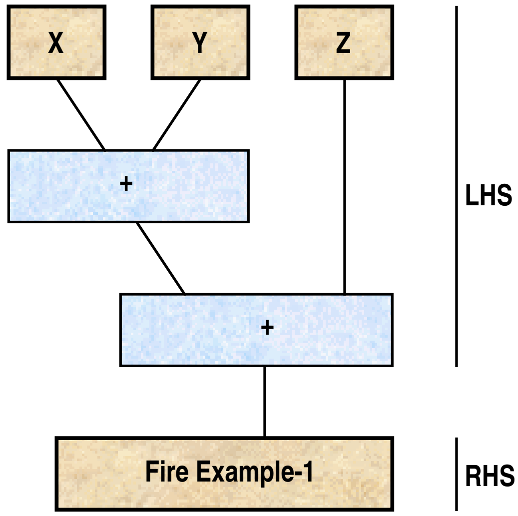

FIG. 1. A rule that would be programmed in CLIPS.[7] Once the facts X, Y, and Z occurs, and satisfy the LHS statement of (X+Y)+Z, then the RHS will fire. Often, the RHS will perform some operation and/or declare a new fact.

Before the monitoring system can perform any sort of reasoning functions, it must first be provided with facts. Using Java Servlet technology,[3] JSDK2.1 allows DAM to run as a servlet and receive information from an Hypertext Transfer Protocol (HTTP) post request. This allows the system to freely accept messages from any user on any host. To preserve security, DAM is designed to refuse information from any host that has not been previously specified as a valid source of knowledge. Presently, the working system at DIII–D may only acknowledge facts that are received from the gat.com domain. However, any off site collaborators that wished to take advantage of DAM would be able to do so by having their domain added to the list of acceptable sources. This makes DAM easily expandable for any and all collaborators, while both centralizing information and relaying information worldwide.

Information from DAM, is sent to the clients through the server push method. Using this method, HTML can be dynamically created, and sent to each client when a new fact in declared. With server push, http connections are held open for an indefinite amount of time. By keeping the connection to clients open, the server is able to send information to each client as it becomes available. The obvious advantage of server push is that DAM can report information in real-time without reestablishing client connections. Unfortunately, server push is not currently compatible with Internet Explorer. Instead of replacing the old HTML web page with an updated version, server push will cause Internet Explorer to append the client’s display, making posted data hard to read. However, server push is fully compatible with Netscape, and provides the capability for clients to receive information in real-time without sending a second request to the server.

DAM itself is essentially composed of three layers, the servlet, the implementation of Rete, and the rules defined in the C Language Integrated Production System (CLIPS) (Fig. 2). Once the DAM Servlet obtains a new message, the corresponding fact is relayed to Rete. The rules to which the algorithm apply the new facts, are defined in the CLIPS language[4] and are evaluated when the new facts are declared. If the LHS of any rule is satisfied, the RHS of that rule is executed. Often, this means a new fact is declared and sent back to the DAM Server to trigger further checks.

SHOT 107832 DATE:2001-06-26				
Time Name	Status	Host	Code	Logfile
12:30:23 ZIPFIT_DONE	ok	star1.gat.com	ZIPFIT	/home/peng/log/zip107832.log
12:28:53 SPECTROSCOPY::TOP.VB.ZEFFQUICK:LOADER	ok	nitron	load_zeffquick	/f/mdsplus/log/autoload/1078/32/ZEFFQUICK_107832.log
12:28:51 CERQUICK_DONE	ok	star1.gat.com	CERQUICK	/home/peng/log/107832.log
12:26:33 SPECTROSCOPY::TOP.PHD.D_ALPHA_FAST:LOADER	ok	ulam.gat.com	load_phd	/f/mdsplus/log/autoload/1078/32/D_ALPHA_FAST_107832.log
12:25:35 IONS::TOP.NEUTRONS.RATES_0D:LOADER	ERROR	nitron	load_neut0d	/f/mdsplus/log/autoload/1078/32/RATES_0D_107832.log
12:25:20 ZIPFIT_ELECTRON_DONE	ok	star1.gat.com	ZIPFIT	/home/peng/log/zip107832.log
12:24:32 EFIT01	ok	lsf002	mdsefit_auto	/f/mdsplus/log/autoload/1078/32/EFIT01_107832.log
12:24:09 TOP.D3DRDB:DBLOADER1	ok	nitron	db_update_basic	/f/mdsplus/log/autoload/1078/32/D3DRDB_107832.log
12:24:05 NEUTRALS::ASDEX:LOADER	ok	nitron	load_asdex	/f/mdsplus/log/autoload/1078/32/ASDEX_107832.log
12:23:57 NEUTRALS::GASFLOW:LOADER	ok	hera.gat.com	load_gasflow	/f/mdsplus/log/autoload/1078/32/GASFLOW_107832.log
12:23:39 SPECTROSCOPY::TOP.PHD:LOADER	ok	ulam.gat.com	load_phd	/f/mdsplus/log/autoload/1078/32/PHD_107832.log
12:22:47 SPECTROSCOPY::TOP.SPRED.MAIN_LINES:LOADER	ok	nitron	load_spred	/f/mdsplus/log/autoload/1078/32/MAIN_LINES_107832.log
12:22:45 EFIT01_FILE	ok	star1.gat.com	EFIT	
12:21:53 ELECTRONS::TOP.CO2:LOADER	ok	badger.gat.com	load_co2	/f/mdsplus/log/autoload/1078/32/CO2_107832.log
12:21:26 TOP:LOADER	ok	nitron	current_shot	/f/mdsplus/log/autoload/1078/32/TOP:LOADER_107832.log
12:16:53 STARTSHOT	ok	atlas.gat.com	d3dinit	

FIG. 2. Information is sent to DAM, where it is examined by the DAM Servlet. If valid, then the Rete algorithm processes the information, and evaluates rules defined in CLIPS. The results are logged to a relational database, and displayed to all clients.

Any fact that is declared, whether from an acceptable foreign host, or from the results of the Rete algorithm, immediately triggers the generation of new HTML for client viewing (Fig. 3). Also, each declared fact is logged to a relational database. At DIII–D, DAM records facts in the D3DRDB[5] relational database so that they can be surveyed in the same way as all the scalar highlights of the DIII–D pulse archive. DAM itself takes advantage of the database logging by querying the database when a user first connects to DAM. If a client connects to DAM mid-day, then DAM will query the database, and the user will still receive all posted information that has previously been obtained for prior pulses.

At DIII–D, DAM (http://nssrv1.gat.com:8000/dam) has been implemented on a Intel PIII 733 MHz machine running Linux Red Hat 6.2. With 1 GB of RAM, and a 100 BaseT network connection into the DIII–D backbone, the machine is capable of running DAM, as well as serving all Unix-run Netscape sessions. Jess 5.2 was installed and integrated in order to provide DAM with an expert system shell. For database connectivity at DIII–D, DAM uses Java's JDBC[3] and Sybase's[6] free dblib client for Linux. DAM has also been developed using Java 1.3 and JSDK2.1.

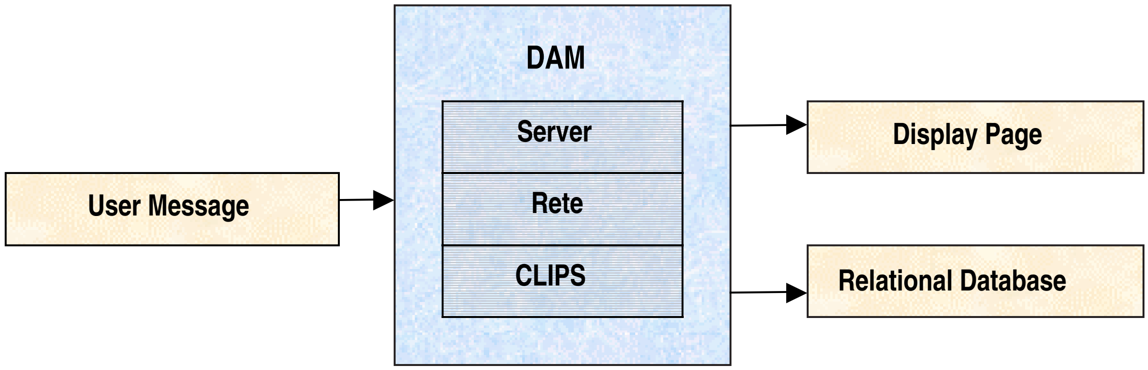

FIG. 3. The HTML user interface for DAM is updated in real-time, as information becomes available HTML links to logfiles provide more detailed information about the reporting process.

III. MONITORING THE BETWEEN PULSE SOFTWARE INFRASTRUCTURE

In general, after a pulse occurs, raw data is acquired, different analysis programs are run, and the resulting analyzed data is centrally stored (Fig. 4). Upon, completion of analysis programs, DAM is notified of a success or failure. This means that DAM is provided with information about the status of a specific analysis code. In other words, DAM would be notified of an error, if there is a problem at any point between the examination of the raw data by a analysis code and the storage of its resulting analyzed data.

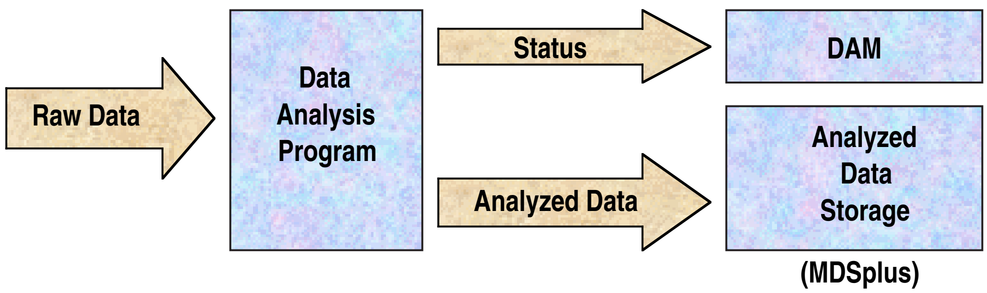

FIG. 4. When the raw data is acquired, a data analysis program is run that produces analyzed data and stores it in a data storage system. DAM is then made aware of the status of the data analysis program.

At DIII–D, when a pulse happens, raw data is acquired on various data acquisition computers. When this raw data is acquired, data analysis routines are dispatched. These routines analyze the raw data, and then store the resulting analyzed data in MDSplus.[7] The loading of analyzed data into MDSplus may be executed on different machines within the UNIX analysis cluster. While the MDSplus loading system is distributed to multiple computers at DIII–D, all machines can report to DAM using the HTTP post operation via a simple Java program. Since most of the MDSplus loading at DIII–D is controlled by a single generic IDL[8] routine, it was only necessary to integrate this Java program with a small number of codes. This enables DAM to track not only the loading of analyzed data into MDSplus, but the status of the analysis program itself.

An example of a dispatched analysis code tracked by DAM, is EFIT. EFIT calculations are conducted in parallel on the “STAR” Linux cluster at DIII–D.[9,10] DAM is used not only to monitor its progress, but also to monitor its performance. To do this, EFIT communicates to DAM when its calculations are complete and also when EFIT data has been successfully loaded into MDSplus. Based on experience running EFIT on the STAR cluster, the amount of time each of these steps should take can be accurately estimated for any plasma pulse.[9,10] If EFIT does not start, complete, or load within the expected time interval, then the monitoring system alerts the users to the problem. In order to accommodate this, DAM has been setup with a timer, so that rules may be written in CLIPS that can report errors referencing data acquisition and loading times.

Since the monitoring system is capable of providing timing information, it would be trivial to track the time interval from the point of acquisition to the point in which analyzed data loading is complete. This would provide a useful measurement of the analysis process for any code. With information about both the raw data and the analyzed data, DAM would be capable of reporting the total time from the acquisition of raw data, to the time the analyzed data was successfully loaded and stored. Once these times are logged, it would be possible to evaluate the efficiency and performance of the entire analysis process, making it possible to identify potential improvements.

IV. PHYSICS ERROR DETECTION

The Data Analysis Monitoring System can also be used to provide error detection in specific physics analysis codes. Results of the error detection can be reported to DAM, alerting a physics team to potential problems that warrant further inspection. For example, DAM currently monitors a comparison between the experimentally measured neutron rate and the expected neutron emission. The measured neutron rate is determined by neutron scintillators, while the expected rate, RDD0D,[11] is calculated by a simple zero-dimensional code that uses central parameters to predict the neutron emission. A comparison algorithm has been created to perform the error checking and reporting to DAM.

At the time RDD0D was designed, steady state neutral beam heating was assumed. The comparison algorithm takes this into account, by comparing the calculated and measured data at the largest time interval of steady state beams. If a pulse does not have a time interval of over 100 ms for which a comparison can be made, then DAM is made aware of this. If there is a time interval to compare the calculated and measured data, then the measured data is averaged over a 10 ms period in order to place it on the same time base as RDD0D. Once this is done, it is possible to compare the measured data to the expected value and produce a ratio for each time point within the valid time interval. From these ratios, it is possible to determine the systematic error for the current pulse. Based on results found in Ref. [11], the calculated neutron rate should be within 30% of the measured rate. If the systematic error lies outside the acceptable limits, then DAM will display an error.

This comparison was tested for all shots during June 2001 (Fig. 5). During June, roughly 18%, or 80 pulses, had an acceptable time window (100 ms) for comparison. For those pulses that were compared, roughly 25% fell outside the acceptable limit, and roughly 75% fell within the allowable error limit. Those pulses that reported an error over 30%, would notify DAM and provide the users with the necessary log file.

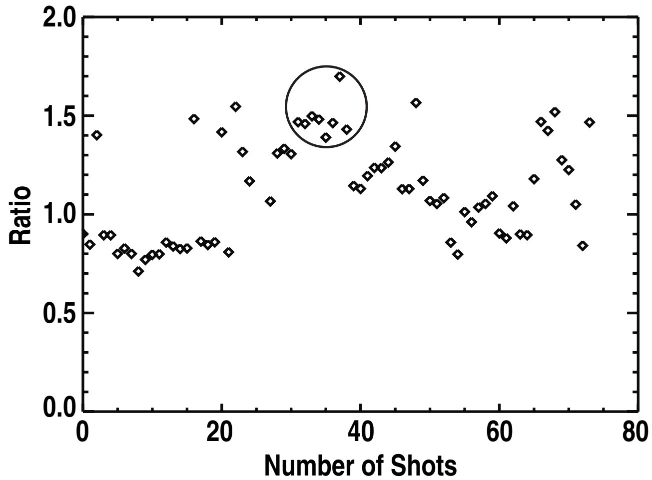

FIG. 5. A Comparison between RDDOD and PLASTIC from June, 2001. Roughly 75% of the pulses fell within acceptable limits. June 6th reported several consecutive pulses with an error larger than the 30%.

Discrepancies can exist either because of a diagnostic failure or because of plasma physics that invalidates the comparison. For example, all pulses on June 6 have RDD0D values that consistently fall outside the acceptable 30% error limit. On this day, significant methane puffing was being performed and therefore the deuteron density was significantly less than the electron density. Since RDD0D assumes deuteron density equals the electron density, on days without impurity injection, such an error detection can quickly alert the physics team to a previously unknown impurity source, which can be addressed during operations. In a similar fashion, if Alfvén eigenmode instabilities are ejecting fast ions from the plasma, RDD0D will be using an overestimate of the fast ion density resulting in an overestimate of the neutron emission. In such a case, DAM will also report an error, alerting the physics team to a problem that requires further analysis and investigation.

Connecting an individual analysis code to DAM is simple. Essentially, if a comparison algorithm can be designed, then it can be connected to DAM. This includes one-to-one comparisons, such as the comparison algorithm for the RDD0D data, as well as more complex algorithms that may even contain multi-dimensional data. Once the comparison algorithm is designed, it merely needs to declare a fact by sending DAM an HTTP post request. Connecting individual physics code to the monitoring system provides several benefits. First, it allows a physicist to track which shots the analysis is working for and for which it fails. This assists the process of discovering new errors and improving the physics code. Another major benefit is that DAM can assist physicists and operators in discovering discrepancies that would otherwise take several pulses, or days to find. Not only does this improve data analysis, but if a discrepancy is discovered that can be quickly fixed, then it increases the number of pulses that can be used for further analysis.

V. DISCUSSION

During the next operation period at DIII–D, DAM is expected to be an integral part of the between pulse software analysis infrastructure. DAM is already used to log the progress of the automatic analysis system during operations, but there is already interest in expanding its use to include time-based rules for each code that the MDSplus launches. To implement this, it is first necessary to compile enough information in the database so that the average completion time of each code can be estimated. With this information, it will be possible to refine the real-time tracking of the MDSplus automatic analysis software, improving both performance and efficiency.

With an increasing amount of programs being connected to DAM, the user interface can quickly become cumbersome. In present use at DIII–D, only a couple dozen facts get sent to each user display for each pulse. However, with even twice as many, for multiple shots per day, the HTML display can quickly become quite long, and overwhelming. In this case, the user interface would need to be redesigned without sacrificing the use of HTML and the benefits of server push.

As MDSplus is being more widely used in the fusion community, it becomes more important to track the automatic dispatchment of anaylsis codes. However, MDSplus does not currently include a software monitoring system. Without creating dependencies between DAM and MDSplus, this new system can be used to provide the software tracking and monitoring capabilities that MDSPlus, and other software infrastructures, are currently lacking.

REFERENCES

[1]Ernest J. Friedman-Hill, "Users Guide for Jess, The Java Expert System Shell," Sandia National Laboratories (2001). (http://herzberg.ca.sandia.gov/jess/).

[2]Charles L. Forgy, Artificial Intelligence **19**, 17-37 (1982).

[3]Sun Microsystems http://java.sun.com.

[4]J.C. Giarrantano, "CLIPS User's Guide," (1998). (http://www.ghg.net/clips/CLIPS.html).

[5]J.R. Burruss, et al., Bull. Am. Phys. Soc. **45**, 280 (2000).

[6]Sybase, Inc. http://www.sybase.com.

[7]J.A. Stillerman, et al., Rev. Sci. Instrum. **68,** 939-942 (1997). (http://www.mdsplus.org).

[8]Research Systems, Inc. http://www.rsinc.com.

[9]Q. Peng, et al., "A Linux PC Cluster for Between-Pulse Magentic Equilibrium Reconstructions and Other Processor Bound Analyses," accepted for publication in Rev. Sci. Instrum.

[10]Q. Peng, et al., "Status of the Linux PC Cluster for Between-Pulse Data Analyses at DIII–D," Presented at the 3rd IAEA Technical Committee Meeting on Control, Data Acquisition, and Remote Participation on Fusion Research, Padova, Italy, July 16-19, 2001, to be published in Fusion Eng. and Design.

[11]W.W. Heidbrink, et al., Rev. Sci. Instrum, Vol. **68**, No.1, 536-539 (1997).

ACKNOWLEDGMENT

The authors would like to thank J.R. Burruss, W.W. Heidbrink, and M.L. Walker for useful discussions. Work supported by U.S. Department of Energy under Contract DE-AC03-99ER54463.