Improvement of Automatic Physics Data Analysis Environment for the LHD Experiment

journal or publication title	Fusion Sicence and Technology
volume	74
number	1-2
page range	161-166
year	2018-03-07
URL	http://hdl.handle.net/10655/00012825

doi: 0.1080/15361055.2017.1390387

Improvement of Automatic Physics Data Analysis Environment for the LHD Experiment

M.Emoto[a], C.Suzuki[a], M.Yokoyama[a,b], M. Yoshinuma[a], R. Seki[a] and K. Ida[a,b]

[a]*National Institue for Fusion Science, 322-6 Oroshi, Toki, 509-5929 Japan*

[b]*Sokendai, Hayama, Kanagaga, 240-0193 Japan*

The physical data of the LHD project has been serviced by the Analyzed Data Server system, and approximately 600 kinds of physical data are served. In order to execute simulation programs for the LHD experiment, one must gather sets of physical data. Because the Automatic Analyzed Server (AutoAna) calculates the physical data automatically, it reduces the scientist's task to collect these physical data. The AutoAna has provided better computing environments for the scientists. However, having recognized its benefits, various requests and issues arise. In this paper, the authors introduce the current status of the AutoAna system.

Keywords: Automation, data handling, data analysis.

I. INTRODUCTION

Typical LHD plasma experiments are executed as repetitive short-pulse experiments. These last for a relatively short time, for example, 10 seconds or less. And plasma discharges are repeated in a 3-minute cycle. Between discharges, the researchers modify experiment conditions. In order to execute the experiment efficiently, they must analyze the latest results between discharges in order to use the results for the successive experiments. Short pulse repetitive operation is common for other fusion experiments, and various systems are used for between-pulse analysis.[1,2] For example, the JET facility, uses Chain1 infrastructure to achieve intershot analyses. The system produces a process chain to run analysis programs synchronizing with the shot sequence. For the DIII-D experiment, MDSplus[3] is situated as the center repository and runs analysis programs automatically using event and dispatching function of MDSPlus. The analysis program waits for an MDSPlus event. If the data is modified or registered, the dispatcher then sends the event to notify the analysis program.

For the LHD experiments, the Automatic Analyzed Server (AutoAna)[4] has been developed. Most of the physics data of the LHD experiment has been managed by the Analyzed Data Server system[5], and the total number of the physical data files has increased to more than 7 million. Some kinds of physical data are calculated from the raw signal data directly, but others are calculated from other physics data. In order to obtain such physics data, dependent data must be available in advance. Furthermore, physical data must be recalculated when the dependent data is updated. In order to launch the calculation program automatically, and to maintain the dependencies among the data, the authors have developed the AutoAna. During the consecutive short pulse experiments, the AutoAna provides the latest data soon after the discharge ends by executing the analysis program automatically. The AutoAna also provides the necessary data to run simulation programs, such as transport analyses. To assemble data to execute simulation program used to be a difficult task for the scientists. Another important task of the AutoAna is to keep the consistency of the physical data. For example, the AutoAna maintains more than 200 kinds of physical data. Most of the data directly or indirectly depend on the temperature profiles mapped on the normalized coordinate. The mapping table is calculated using the measured data of Thomson scattering diagnostics and magnetic fields data. The mapping table is changed sometimes, because the algorithm of the mapping has been improved and the magnetic field database is expanding. Therefore, it is often the case that nearly 200 kinds of physical data, or several million physical data files must be recalculated. Because the AutoAna automatically updates when the temperature profile is updated, it eliminates the recalculation task, and the scientists can refer to the up-to-date data.

Because of the historical reasons, the computer system of the LHD is not centralized compared to JET or DIII-D. Various user computers are distributed around the network, and they engage in data acquisition and analysis. Therefore, data registration and update are not scheduled, the system cannot anticipate which data will come next, and tens of thousands of data may be updated in a very short time. Therefore, the AutoAna system must be flexible to provide the necessary information for the experiment coordinator to decide the experiment parameters within the 3-minute cycle even when a sudden request for calculation arises. In order to satisfy this requirement, the AutoAna is built as a network distributed system, and it can easily increase its performance by increasing the number of computers.

In this way, the AutoAna has contributed the useful data analysis environments to scientists. However, having recognized the utility of the AutoAna, various requests have appeared. One of them was to run complex analysis programs that took a very long time to complete the calculations. Because such programs prevent other existing programs from running, it was impossible to provide the latest summary results during the short pulse experiments while these programs were running. Another request was to provide the user-friendly interface to view the status of the programs, and to run them manually. In this paper, the recent improvements for overcoming these difficulties and the outlook for the future plans of the AutoAna will be described in detail.

II. AUTOMATIC ANALYSIS SYSTEM (AutoAna)

II.A OVERVIEW

Fig. 1 shows the overview of the AutoAna. The AutoAna consists of three major components, Server, Updater, and Executer. These components are written in Ruby, and communicate with each other using dRuby library package. They manage the analyzed programs as modules. The module defines the source physical data, output physical data, program command, and others. The module requires zero or more input data. If the module does not require input data, the program is executed only manually. The module produces at least one output data, and a different module does not produce the same output data. Fig. 2 shows an example of dependency diagram of the modules and physical data. This figure shows the flow to calculate a physical data “dytran7”, dynamic transport analysis, using “lhdcxs7_cvi”, carbon line data obtained by charge exchange spectroscopy. The module, “dytrans7”, uses the physical data “cxsmap7”. This data is the distribution of carbon line on the common coordinate calculated from real coordinate distribution of “lhdcxs7_cvi” using the mapping table, “tsmap”. “tsmap” is calculated from “thomson”, electron temperature distribution measured by Thomson scattering measurement, and “fir_nel”, line integrated electron density measured by FIR. The actual definition of the modules is described in a single JSON file as shown in Fig.3. Fig.3 corresponds to Fig.2. A module “mapping_ts” requires “ip”, “thomson”, and “fir_nel” as input, and produces “mapping_ts” using a command “mapping_ts.sh”. Another module, “tsmap”, uses “mapping_ts” and three other physical data as input to produces “tsmap”, and other physical data. When a new physical data is registered to the Analyzed Data Server, the server sends the notification of new data registration by IP multicast. IP multicast is sent to all the clients who want the same data by a single transmission. Therefore, it is efficient to send the same data to multiple clients. Receiving the registration notification packets, the client can know that new data is registered, and can obtain the latest data. When Updater receives the IP multicast packet, it checks if there are modules that require the newly registered physical data as input source and checks if all the input data is prepared or not. If such modules are found, Updater asks Server to add a new job to the queue. The queue is implemented as FIFO (First In First Out). Executer retrieves a job from the queue on a first come, first served basis, and Executer runs the script of the module in order to produce output physical data. Because multiple Executers are running, multiple analysis programs are working simultaneously to enhance performance.

II.B NEW REQUIREMENTS

The AutoAna has contributed to provide the researcher a useful simulation environment. Previously, in order to execute simulation, the researcher had to ask many people to provide their data. Therefore, the simulation was executed only for limited number of discharge experiments. Currently, necessary data is registered by the AutoAna automatically, in principle simulation can be executed for all of the plasma experiments. Having recognized its utilities, various requests arose. In order to reply to these requests, the authors have improved the AutoAna as follows.

a) To run the program that requires a very long time

The average time to run the analysis program is from about ten seconds to tens of minutes using one CPU core. In order to catch up with the 3-minute plasma discharge cycle, these processes are executed simultaneously, and currently more than 100 processes are executed at once. Although the calculation of some physical data cannot be finished by the next plasma shot, physical data can be available by several shots later. However, the run-time of the newly requested program is about several hours using 28 CPU cores. Executing such a time-consuming program disturbs the jobs of other programs, and the AutoAna cannot catch up with the cycle.

In order to solve this program, an extra queue has been introduced. The programs that require much time are registered in to the extra queue instead of the standard queue and do not disturb execution of jobs in the standard queue. The job in the extra queue has a priority, and the higher priority job starts prior to the lower job even if the higher priority job is registered later.

b) User-friendly interface to view jobs and results and to run the program manually

Due to various reasons, physical data fails to be registered. For example, there is a lack of source data, data format error, many jobs waiting to run, and others. Therefore, users want a user-friendly interface in order to know the situation so to solve problems. There is also the requirement to run the module manually. When resolving the failure, the failed job must be submitted. However, the AutoAna does not know whether the problem has been fixed or not. Therefore, the program must be run manually. There is another reason to run a program manually. If there is a job that requires many computer resources and disturbs the execution of other jobs, it should usually be disabled and executed by request.

In order to meet the request, a new property “enabled” in the module description file is introduced. If the property “enabled” is true, the module is executed by the AutoAna. If the property “enabled” is false, the module is disabled, and it is executed manually. Also, Web interface has been developed. Fig. 4. shows this interface. Accessing this web page, the user can see the status of the modules in colors. Green indicates the target physical data is registered, orange indicates the physical data is not registered because not all the input data is available, and red indicates the physical data is not registered even though all the input data is available. Yellow indicates the physical data is registered but it is older than input data. This situation occurs when the input data is recalculated, but the dependent data is not updated yet. Clicking the target shot number, a pop-up of the detailed information appears. The user can know what happened to the module, and submit the jobs from this pop-up.

c) Run programs that require limited resource

The analysis program is to run receiving a shot number as an argument, calculates the target physical data, and registers it to the Analyzed Data Server. Generally, analysis programs are offered by the scientist who specializes in their fields. Therefore, the programs are written in various languages, for example, Python, FORTRAN, PV-Wave, Shell script, and others. There is also a potential requirement to use other proprietary tools such as IDL and MATLAB because both tools are widely used as data analysis tools in the nuclear fusion community. However, both tools were not supported because the number of run-time license is limited. As previously mentioned, execution programs run simultaneously. The number of concurrent processes is 116 at most, but the number is easy to increase by adding Executer PC to the AutoAna and consumes limited licenses. In order to make use of the limited license, a resource management scheme is introduced. Fig. 5 shows the procedure to manage computer resources. All the available resources and the resources that the module uses are written in a new JSON file, “resource.json”. In this example, there are 10 licenses for IDL and 5 licenses for

MATLAB. Then, a new tag “USE” is added in the module definition file. In the example, “module 1” uses 1 IDL license. When Executer run “module 1”, it asks ResourceManager if an IDL license is available or not. If the license is available, it rents one license from ResourceManager. When the execution is terminated, it returns the license to ResourceManager.

d) **Manage the programs by the user**

The analysis programs are offered by researchers who specialize in their fields, and the programs should be used as they are. However, the programs cannot be used without modification by the following reasons.

- The programs are not assumed to run simultaneously. They use the same temporary file, and conflict with other processes.
- They do not clean the temporary files, and it causes disk shortage.
- They are not considered to run in different environments from the creator. For example, they use the absolute path, user defined environment variables, and others.

However, because the analysis programs are being improved, the researchers want to manage the programs by themselves. Currently, this problem is not solved yet, but it is discussed in the next section.

III. DISCUSSION AND FUTURE PLANS

The goal of AutoAna is to provide the complete physical data set. Scientists can easily replace their analysis program in order to reflect their new study, and all the dependent data is quickly recalculated by replacement of the programs. In order to allow the researchers to replace their program by themselves, the program must be run in an isolated environment. Effects of the program are restricted in the isolated environment, the programmer does not need to care for the behavior of the programs. For example, the temporary files are not altered by other programs running outside the isolated environment. To realize an isolated environment, virtualization technology is often used. The virtualization technology can be classified by the level of virtualization. Full verticalization, such as VMWare and Xen, simulates a hardware to run the operating system. On the other hand, operating-system-level virtualization, such as virtuozzo[6] and Docker[7], simulates the operating system to run the isolated program. The isolation level of the former is higher than the latter but is less efficient because it simulates the hardware. In order to gain CPU performance, the AutoAna needs to make the best use of CPU resources, and operating-system-level virtualization is preferable. At present, Docker is the most promising candidate. Using a Docker hub, the programmers can easily build their own running environment from a templates environment.

In order to achieve quick updates of all the data, increase of performance is imperative. To gain the calculation performance, the number of computers to run Executor has been increased, and currently more than 100 processes at most are running simultaneously. However, processes access the same resources simultaneously, such as the database server and the file server. This causes a concentration on access. Therefore, it has become difficult to increase total performance just by increasing the number of computers. In order to increase total performance of the AutoAna, multiple server configurations as well as the improvement of the network infrastructure must be considered.

ACKNOWLEDGEMENTS

The authors are grateful to all the staff of the LHD for this opportunity to develop this system. This work is supported by the National Institute for Fusion Science (budget No.: NIFS10ULHH014, ULII001).

REFERENCES

1. R. LAYNE, N. COOK, D. HARTING, D.C. MCDONALD, C. TIDY, “The JET Intershot Analysis: Current infrastructure and future plans”, *Fusion Eng. Des*, **85** 403 (2010)

2. D. P. SCHISSEL, G. ABLA, S. FLANAGAN, L. KIM, AND X. LEE, ‘The Between-Pulse Data Analysis Infrastructure at the DIII-D National Fusion Facility’, *Fusion Sci. Technol.*, **58** 720 (2010)

3. J. A. STILLERMAN, T.W. FREDIAN, K. A. KLARE, AND G. MANDUCHI, “MDSplus Data Acquisition System”, *Rev.Sci. Instrum.*, **68**, 939 (1997)

4. M. EMOTO, C. SUZUKI, Y.SUZUKI, et al., “Performance Improvement in Real-Time Mapping of Thomson Scattering Data to Flux Coordinates in LHD”, *J. Plasma Fusion Res.*, **7**, (2012) 2405058

5. M. EMOTO, S. OHDACHI, K. WATANABE, et. al., “Server for experimental data from LHD”, *Fusion Eng. Des*, **81** 2019 (2006)

6. http://www.parallels.com/

7. https://www.docker.com/

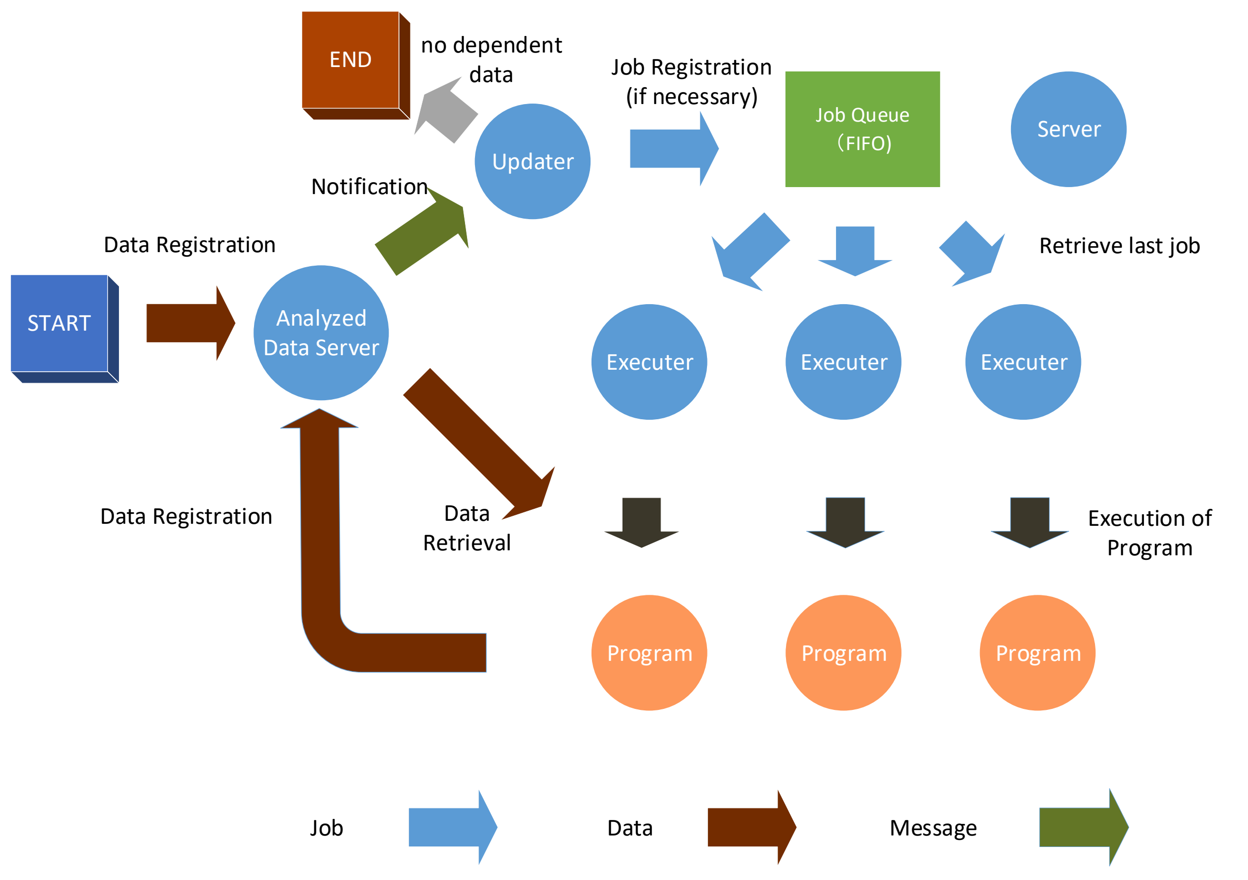

END
no dependent data
Job Registration (if necessary)
Job Queue (FIFO)
Server
Updater
Notification
Data Registration
START
Analyzed Data Server
Retrieve last job
Executer
Executer
Executer
Data Registration
Data Retrieval
Execution of Program
Program
Program
Program
Job
Data
Message

Fig. 1

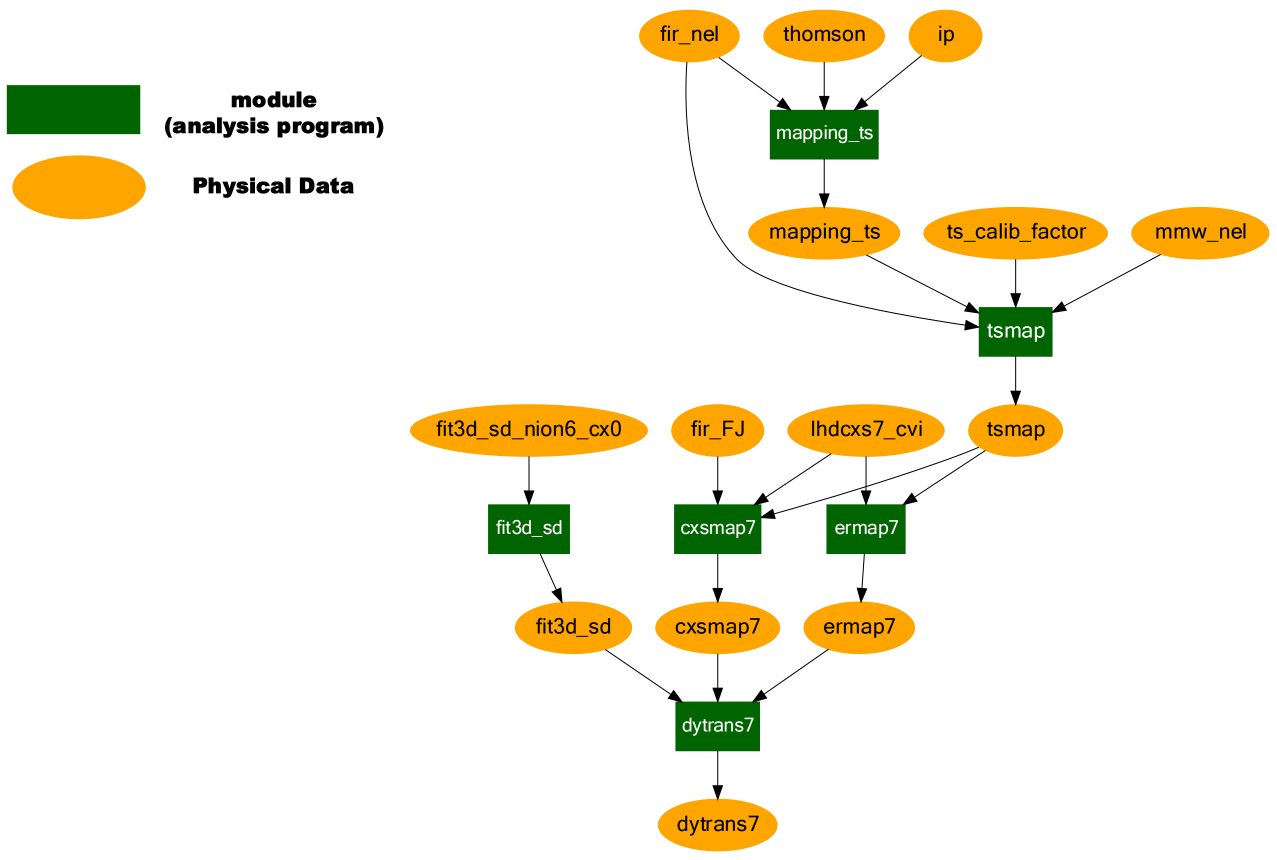
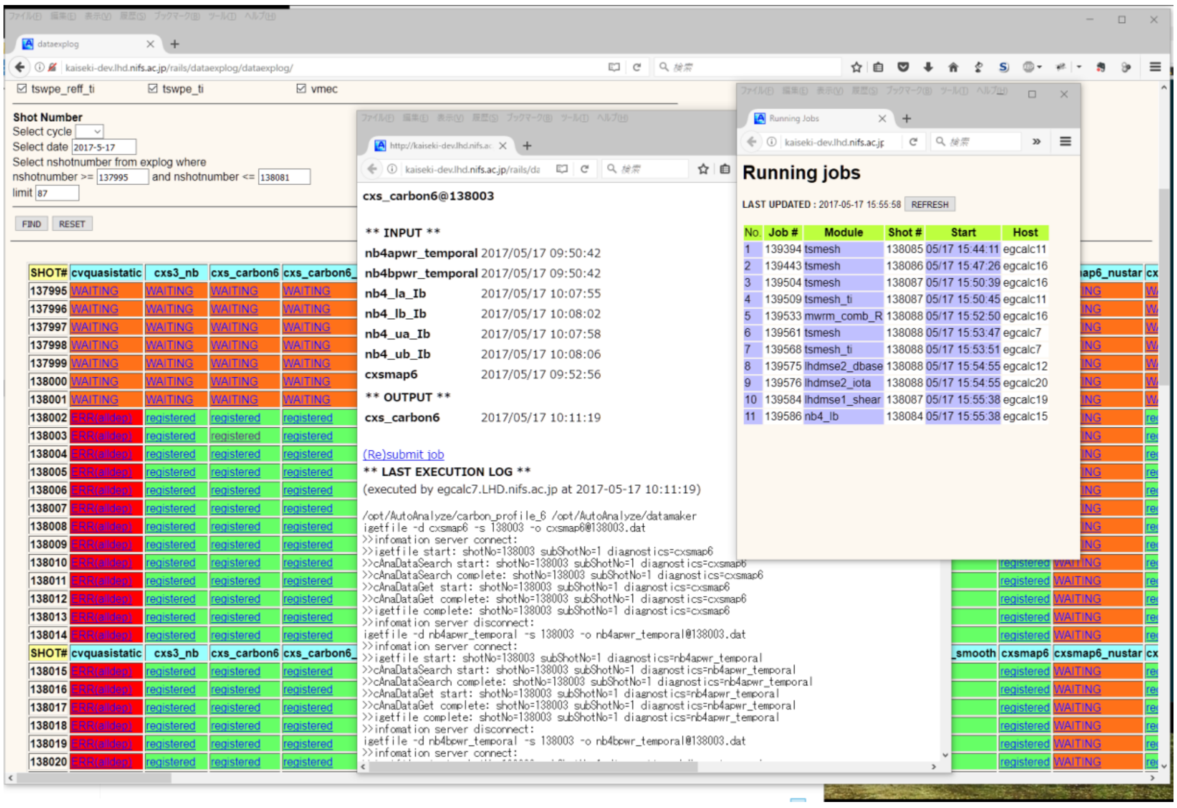

module (analysis program)
Physical Data
fir_nel
thomson
ip
mapping_ts
mapping_ts
ts_calib_factor
mmw_nel
tsmap
fit3d_sd_nion6_cx0
fir_FJ
lhdcxs7_cvi
tsmap
fit3d_sd
cxsmap7
ermap7
fit3d_sd
cxsmap7
ermap7
dytrans7
dytrans7

Fig. 2

```
{
  "mapping_ts": {
    "depend": [
      "ip",
      "thomson",
      "fir_nel"
    ],
    "output": [
      "mapping_ts"
    ],
    "enabled" : true,
    "command": "$TOPDIR/tsmap/mapping_ts.sh %d"
  },
  "tsmap": {
    "depend": [
      "mapping_ts",
      "ts_calib_factor",
      "mmw_nel",
      "fir_nel"
    ],
    "output": [
      "tsmap"
    ],
    "enabled" : true,
    "command": "$TOPDIR/tsmap/tsmap.sh %d"
  },
  "cxsmap7": {
  },...
  ...
}
```

Fig. 3



Shot Number
Select cycle
Select date 2017-5-17
Select nshotnumber from explog where
nshotnumber >= 137995 and nshotnumber <= 138081
limit 87

SHOT#	cvquasistatic	cxs3_nb	cxs_carbon6	cxs_carbon6_
137995	WAITING	WAITING	WAITING	WAITING
137996	WAITING	WAITING	WAITING	WAITING
137997	WAITING	WAITING	WAITING	WAITING
137998	WAITING	WAITING	WAITING	WAITING
137999	WAITING	WAITING	WAITING	WAITING
138000	WAITING	WAITING	WAITING	WAITING
138001	WAITING	WAITING	WAITING	WAITING
138002	ERR(alldep)	registered	registered	registered
138003	ERR(alldep)	registered	registered	registered
138004	ERR(alldep)	registered	registered	registered
138005	ERR(alldep)	registered	registered	registered
138006	ERR(alldep)	registered	registered	registered
138007	ERR(alldep)	registered	registered	registered
138008	ERR(alldep)	registered	registered	registered
138009	ERR(alldep)	registered	registered	registered
138010	ERR(alldep)	registered	registered	registered
138011	ERR(alldep)	registered	registered	registered
138012	ERR(alldep)	registered	registered	registered
138013	ERR(alldep)	registered	registered	registered
138014	ERR(alldep)	registered	registered	registered
SHOT#	cvquasistatic	cxs3_nb	cxs_carbon6	cxs_carbon6_
138015	ERR(alldep)	registered	registered	registered
138016	ERR(alldep)	registered	registered	registered
138017	ERR(alldep)	registered	registered	registered
138018	ERR(alldep)	registered	registered	registered
138019	ERR(alldep)	registered	registered	registered
138020	ERR(alldep)	registered	registered	registered

cxs_carbon6@138003

** INPUT **

nb4apwr_temporal	2017/05/17 09:50:42
nb4bpwr_temporal	2017/05/17 09:50:42
nb4_la_Ib	2017/05/17 10:07:55
nb4_lb_Ib	2017/05/17 10:08:02
nb4_ua_Ib	2017/05/17 10:07:58
nb4_ub_Ib	2017/05/17 10:08:06
cxsmap6	2017/05/17 09:52:56

** OUTPUT **

cxs_carbon6	2017/05/17 10:11:19

(Re)submit job

** LAST EXECUTION LOG **

(executed by egcalc7.LHD.nifs.ac.jp at 2017-05-17 10:11:19)

```
/opt/AutoAnalyze/carbon_profile_6 /opt/AutoAnalyze/datamaker
igetfile -d cxsmap6 -s 138003 -o cxsmap6@138003.dat
>>infomation server connect:
>>igetfile start: shotNo=138003 subShotNo=1 diagnostics=cxsmap6
>>cAnaDataSearch start: shotNo=138003 subShotNo=1 diagnostics=cxsmap6
>>cAnaDataSearch complete: shotNo=138003 subShotNo=1 diagnostics=cxsmap6
>>cAnaDataGet start: shotNo=138003 subShotNo=1 diagnostics=cxsmap6
>>cAnaDataGet complete: shotNo=138003 subShotNo=1 diagnostics=cxsmap6
>>igetfile complete: shotNo=138003 subShotNo=1 diagnostics=cxsmap6
>>infomation server disconnect:
igetfile -d nb4apwr_temporal -s 138003 -o nb4apwr_temporal@138003.dat
>>infomation server connect:
>>igetfile start: shotNo=138003 subShotNo=1 diagnostics=nb4apwr_temporal
>>cAnaDataSearch start: shotNo=138003 subShotNo=1 diagnostics=nb4apwr_temporal
>>cAnaDataSearch complete: shotNo=138003 subShotNo=1 diagnostics=nb4apwr_temporal
>>cAnaDataGet start: shotNo=138003 subShotNo=1 diagnostics=nb4apwr_temporal
>>cAnaDataGet complete: shotNo=138003 subShotNo=1 diagnostics=nb4apwr_temporal
>>igetfile complete: shotNo=138003 subShotNo=1 diagnostics=nb4apwr_temporal
>>infomation server disconnect:
igetfile -d nb4bpwr_temporal -s 138003 -o nb4bpwr_temporal@138003.dat
>>infomation server connect:
```

Running jobs

LAST UPDATED : 2017-05-17 15:55:58 REFRESH

No.	Job #	Module	Shot #	Start	Host
1	139394	tsmesh	138085	05/17 15:44:11	egcalc11
2	139443	tsmesh	138086	05/17 15:47:26	egcalc16
3	139504	tsmesh	138087	05/17 15:50:39	egcalc16
4	139509	tsmesh_ti	138087	05/17 15:50:45	egcalc11
5	139533	mwrm_comb_R	138088	05/17 15:52:50	egcalc16
6	139561	tsmesh	138088	05/17 15:53:47	egcalc7
7	139568	tsmesh_ti	138088	05/17 15:53:51	egcalc7
8	139575	lhdmse2_dbase	138088	05/17 15:54:55	egcalc12
9	139576	lhdmse2_iota	138088	05/17 15:54:55	egcalc20
10	139584	lhdmse1_shear	138087	05/17 15:55:38	egcalc19
11	139586	nb4_lb	138084	05/17 15:55:38	egcalc15

Fig. 4

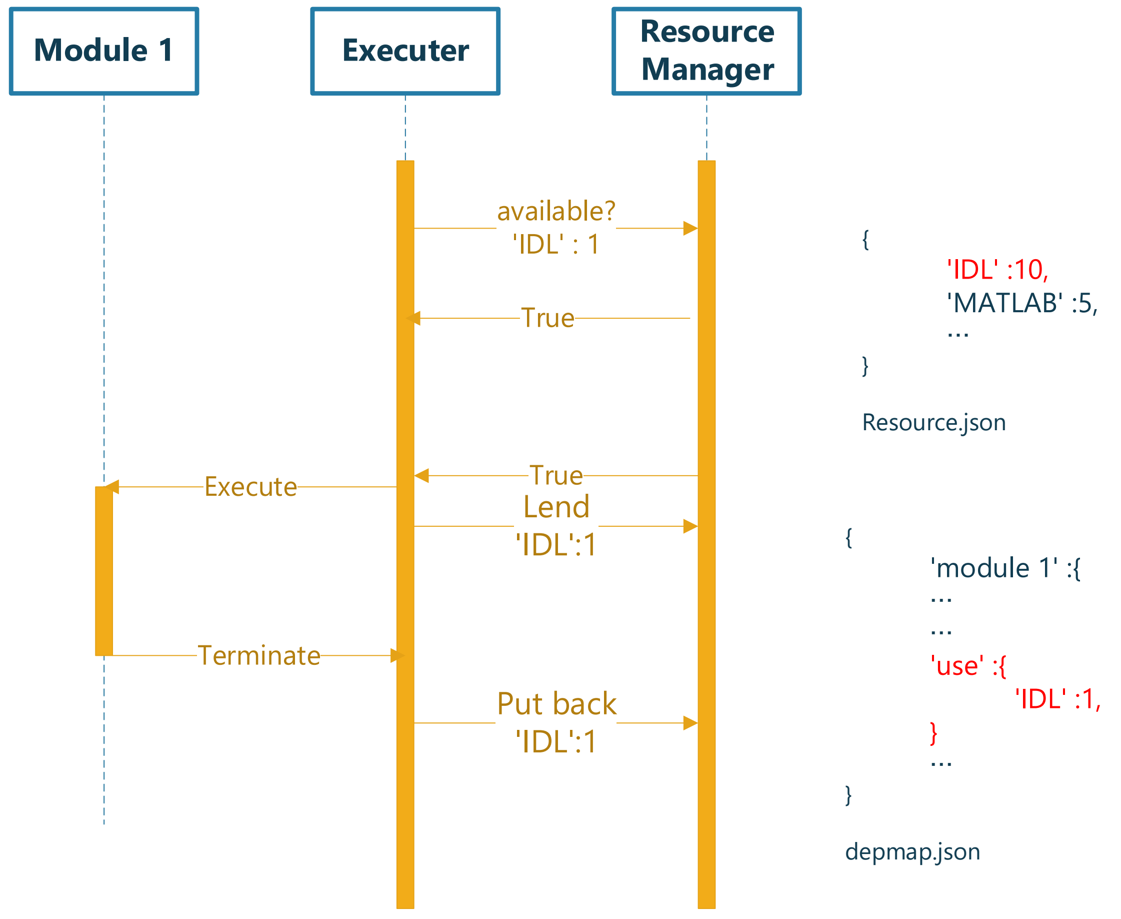

Module 1
Executer
Resource Manager
available? 'IDL' : 1
True
True
Execute
Lend 'IDL':1
Terminate
Put back 'IDL':1
{
'IDL' :10,
'MATLAB' :5,
…
}
Resource.json
{
'module 1' :{
…
…
'use' :{
'IDL' :1,
}
…
}
depmap.json

Fig. 5

Figure Captions

Fig.1. Overview of the Automatic Analysis System. It consists of three major components, Server, Updater and Executer.

Fig.2. A part of dependency diagram of physical data and modules. The rectangle indicates module and the oval indicates physical data.

Fig. 3. Module definition file of Fig.2.

Fig. 4. Web interface for the Automatic Analysis System. The status of the queue is colored. The user can view the detailed status of the queue, and can submit a new job manually.

Fig. 5. This figure shows procedures to obtain necessary computer resources. Executer asks Resource Manager to rent resources. Available resource and the resource that the module uses are written in a JSON file.