Available online at www.sciencedirect.com

ScienceDirect

Fusion Engineering and Design 83 (2008) 453–457

Fusion Engineering and Design

www.elsevier.com/locate/fusengdes

Web interface for plasma analysis codes

M. Emoto [a,*], S. Murakami [b], M. Yoshida [a], H. Funaba [a], Y. Nagayama [a]

[a] *National Institute for Fusion Science, 322-6 Oroshi, Toki, Gifu 509-5292, Japan*
[b] *Kyoto University, Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, Japan*

Available online 21 February 2008

Abstract

There are many analysis codes that analyze various aspects of plasma physics. However, most of them are FORTRAN programs that are written to be run in supercomputers. On the other hand, many scientists use GUI (graphical user interface)-based operating systems. For those who are not familiar with supercomputers, it is a difficult task to run analysis codes in supercomputers, and they often hesitate to use these programs to substantiate their ideas. Furthermore, these analysis codes are written for personal use, and the programmers do not expect these programs to be run by other users. In order to make these programs to be widely used by many users, the authors developed user-friendly interfaces using a Web interface. Since the Web browser is one of the most common applications, it is useful for both the users and developers. In order to realize interactive Web interface, AJAX technique is widely used, and the authors also adopted AJAX. To build such an AJAX based Web system, Ruby on Rails plays an important role in this system. Since this application framework, which is written in Ruby, abstracts the Web interfaces necessary to implement AJAX and database functions, it enables the programmers to efficiently develop the Web-based application. In this paper, the authors will introduce the system and demonstrate the usefulness of this approach.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Web; Ruby; Ruby on rails; AJAX; Analysis code; Super computer

1. Introduction

To analyze various aspects of plasma physics, several analysis codes such as VMEC [1], HINT [2], and TASK [3] have been developed. Most of them are usually written in FORTRAN and run on supercomputers. Many of these codes are command-line applications that must be invoked through terminal emulators, while PC users typically use GUI-based applications with mouse operations. Furthermore, the necessary parameters are usually provided as text files and users must write them manually; this often results in errors.

In order to facilitate the widespread use of the existing codes, a user-friendly interface is required. Therefore, the authors developed Web interfaces for these codes. By using these interfaces, researchers can interactively provide the necessary parameters to the codes through familiar Web browsers. However, because the basic Web services are synchronous, i.e., the browser must stop redrawing until it receives the entire content, they ruin the smooth operation. To solve this problem, the AJAX (Asynchronous JavaScript and XML) [4] technique is widely used. This technique uses JavaScript to asynchronously redraw the Web pages without any pause. The authors also use the AJAX technique to establish a smooth interface. In this system, Ruby on Rails [5] plays an important role. It is an application framework written in Ruby, and the Web services can be provided as Ruby class methods. Because Ruby on Rails provides the base classes to handle AJAX and database functions, it enables the users to efficiently develop the Web-based applications without using JavaScript or SQL.

To demonstrate the usefulness of this approach, the authors have developed a Web interface for the FIT code [6]. The FIT code was developed by Murakami.; it is a simulation code that simplified GNET [7] to study the energetic ion distribution in helical plasma. It runs on supercomputers and requires a text file containing the distributions of electron temperatures, electron densities, and ion temperatures as polynomial expressions of the minor plasma radius. This approach poses difficulties for users who are not involved in the project because they do not know where the data is stored or how to read it. Therefore, for such users, a more user-friendly interface is required for the execution

* Corresponding author. Tel.: +81 572 58 2242; fax: +81 572 58 2624.
E-mail address: emo@nifs.ac.jp (M. Emoto).

0920-3796/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.fusengdes.2007.10.008

of the code. In the following sections, the authors describe this new approach in detail.

2. System design

To increase the utility of the existing analysis codes, the authors developed user-friendly interfaces for the existing codes. The main target users are co-researchers of the LHD (large helical device) at NIFS (National Institute for Fusion Science). They are typically not familiar with either the supercomputers in the NIFS or data-retrieving tools for the LHD experiment; however, they must analyze the data during their short-term visits. In order to realize this interface, the authors developed the following design policies.

2.1. GUI interfaces

For the short-term visitors, even finding the necessary data is an obstacle to the use of the analysis codes. Therefore, the system should use GUI as much as possible so that the necessary information can be easily found. For example, when the analysis code requires an experiment number, the interface should be designed in such a manner that the user can choose the number from a list of candidates instead of entering the number manually.

2.2. Unnecessary to install

The installation of necessary software is another obstacle to the user. Therefore, the authors adopted the Web browser as the front end of the interface. Because the Web browser is one of the most basic applications of the current operating system, the users do not have to install other software. This feature is convenient not only for users but also for developers because they do not have to distribute the software to the users, and it is easy to maintain this software.

3. System overview

Fig. 1 shows the overview of the Web interface for the FIT code. The authors adopted the three-tier model; the system consists of three layers: the presentation layer, business logic layer, and analysis codes. The presentation layer is the Web server. It is the front end of the system, and it directly communicates with the users. The business logic layer is the intermediate layer between the Web server and the analysis codes. It manages the client requests and converts them into a format that is acceptable by the analysis codes. This layer also manages the job queue of the analysis codes and calculation results. The analysis codes are written in FORTRAN and run on the supercomputer.

The reason for the selection of the three-tier model is that it is extensible and easy to maintain; each layer is loosely connected to the other layers, and it can easily be replaced with another component.

3.1. Presentation layer

The parameters required to execute the FIT code are NBI power, electron temperature, electron density, ion density, etc. These data are retrieved from the database interactively using the GUI (Fig. 2). The temperature distribution is plotted on a graph, and the user can confirm its distribution. The GUI is realized by the following technologies.

3.1.1. Extension to HTML

Although the authors adopted the Web browser as the front end of the system, it is difficult to implement a user-friendly GUI only by the basic functions of the Web browser because the functions of HTML are limited. Therefore, other languages such as Java, JavaScript, and ActiveX are required to realize such an interface. The authors chose JavaScript as a supplementary lan-

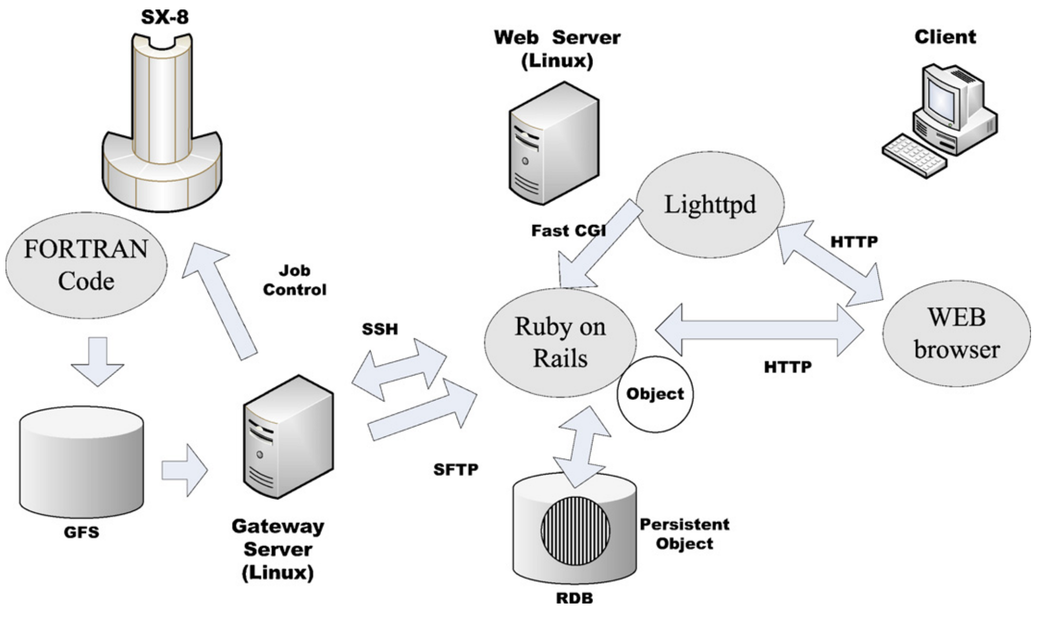
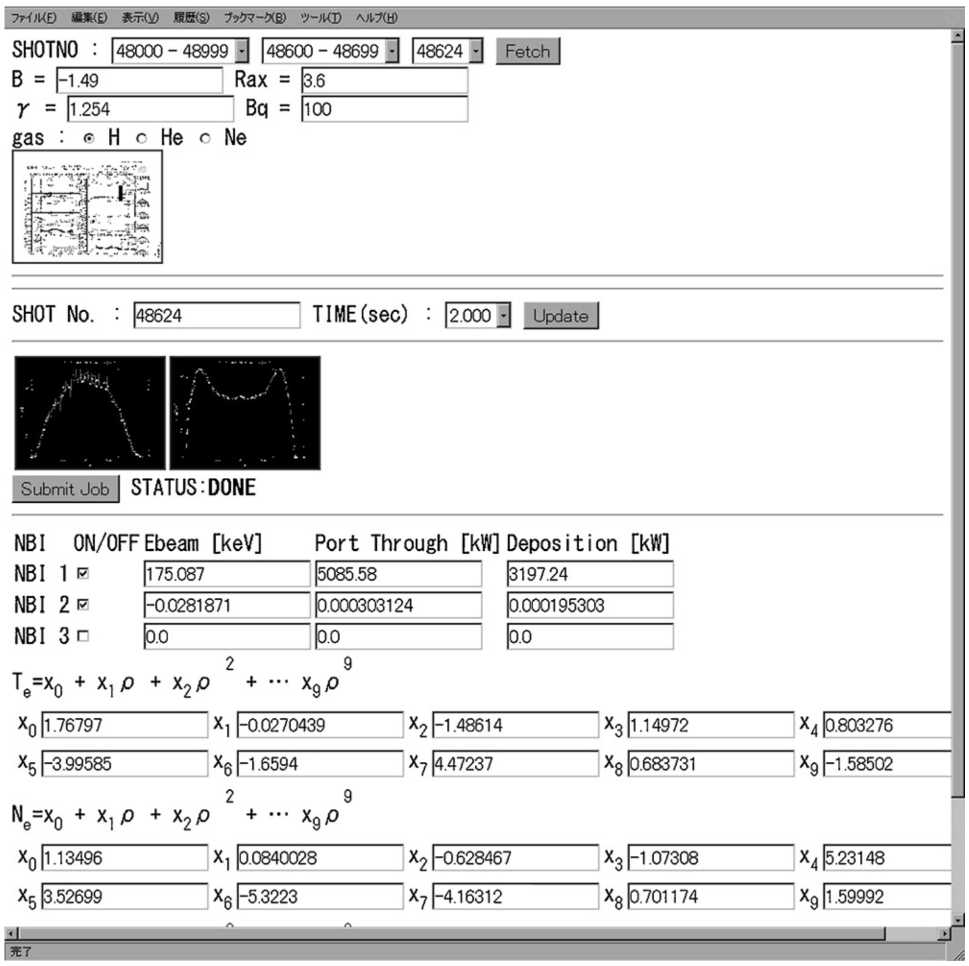


Fig. 1. System overview.


ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H)
Fetch
Update
Submit Job
完了

SHOTNO : 48000 – 48999 | 48600 – 48699 | 48624

B = −1.49 Rax = 3.6

γ = 1.254 Bq = 100

gas : [✓ H] He Ne

SHOT No. : 48624 TIME(sec) : 2.000

STATUS:**DONE**

NBI	ON/OFF	Ebeam [keV]	Port Through [kW]	Deposition [kW]
NBI 1	☑	175.087	5085.58	3197.24
NBI 2	☑	−0.0281871	0.000303124	0.000195303
NBI 3	☐	0.0	0.0	0.0

$T_e = x_0 + x_1\rho + x_2\rho^2 + \cdots x_9\rho^9$

x_0 1.76797	x_1 −0.0270439	x_2 −1.48614	x_3 1.14972	x_4 0.803276
x_5 −3.99585	x_6 −1.6594	x_7 4.47237	x_8 0.683731	x_9 −1.58502

$N_e = x_0 + x_1\rho + x_2\rho^2 + \cdots x_9\rho^9$

x_0 1.13496	x_1 0.0840028	x_2 −0.628467	x_3 −1.07308	x_4 5.23148
x_5 3.52699	x_6 −5.3223	x_7 −4.16312	x_8 0.701174	x_9 1.59992

Fig. 2. Web interface for FIT code.

guage. This is because the major Web browsers such as Internet Explorer and Mozilla Firefox inherently support JavaScript, and the users do not have to install other components.

In comparison with other options such as Java, the functions of JavaScript are limited; for example, it cannot directly connect to the database. However, because the authors adopted the three-their model and the main role of the presentation layer is to provide the GUI, the functions of JavaScript are sufficient for this purpose.

3.1.2. AJAX

The Web pages are written in both HTML and JavaScript; however, the protocol between the client and the server is HTTP. Therefore, from the viewpoint of the client-server relation, it is not different from a normal Web server. The user submits the necessary parameters through the Web interface and calls the programs using CGI (common gateway interface). However, it takes several minutes to execute the FIT code. During the execution, the Web browser must postpone the display of the results page until it receives the results from the Web server. Therefore, a user must wait until the next page loads; this disrupts the flow of the site. This is because the CGI program is “synchronously” called from the browser and the browser postpones the loading of a page until it receives the content. To solve this problem, the authors have adopted AJAX. Instead of waiting for the entire content to arrive, the browser shows a part of the content; when the JavaScript code receives the remaining content from the CGI program, it redraws the page.

3.2. Business logic layer

3.2.1. Ruby on Rails

To implement AJAX, the developer has to combine HTML and JavaScript to show the Web pages. This degrades the readability and maintainability of source codes. Furthermore, because the Web server does not have a common format to exchange complex format values such as structured data, it is the responsibility of the developers to handle complex values. As the name suggests, AJAX often uses XML to encode complex values; however, it requires extra codes to handle XML.

For efficient development, a development tool is required to implement AJAX. The authors adopted Ruby on Rails for this purpose. Ruby on Rails is an application framework, and the Web services are implemented as Ruby class methods. The Web pages are dynamically produced from the Ruby codes, and Ruby on Rails encapsulates the forms of HTML and the JavaScript codes to realize AJAX. Basically, the users can use only Ruby to implement AJAX applications without using JavaScript. Furthermore, the parameters sent from or to the servers can be accessed as Ruby variables; complex variables can be handled easily.

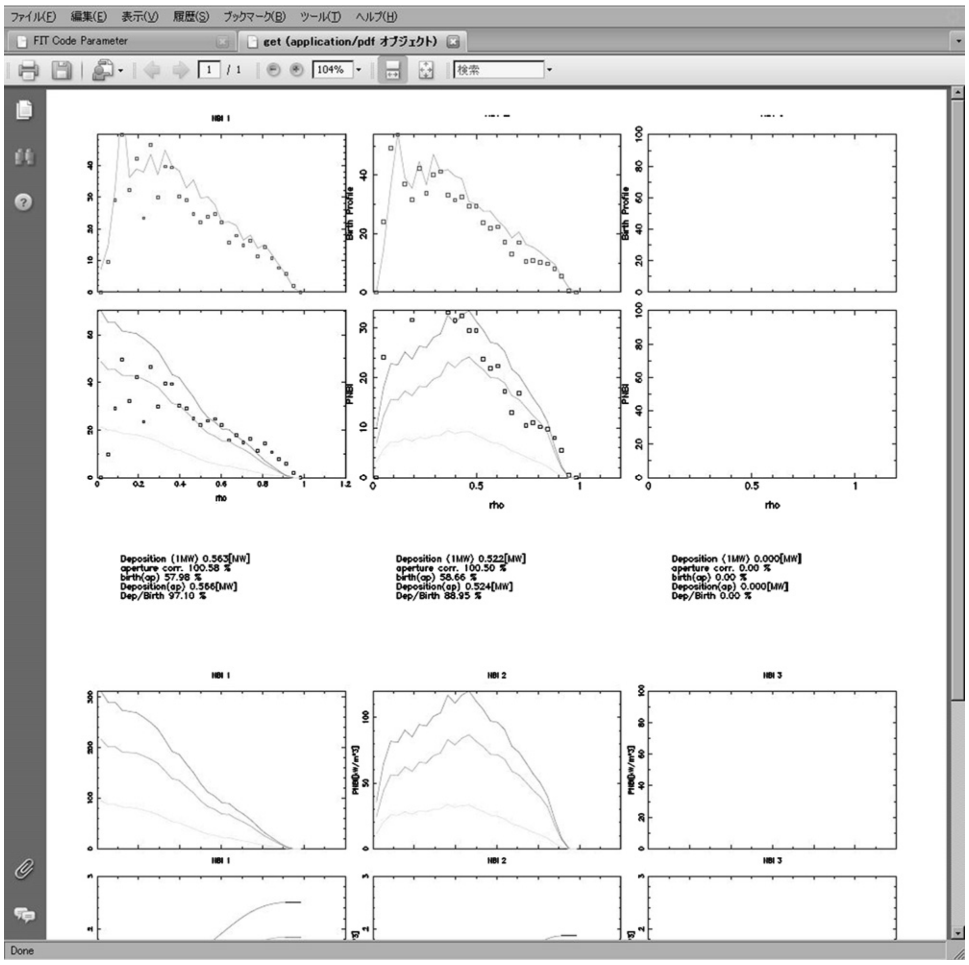
The services of Ruby on Rails are invoked from the Web server as CGI. It takes time to execute the CGI programs because the Web server must create a new process for each request. To reduce the execution time, these applications are run as FastCGI [8]; they continue to run for a while after one request is completed, and then the applications are reused for the next request.

3.2.2. Database

The disadvantage of HTTP is that each request is independent, while the normal network protocol maintains its connection while the server and the client are communicating with each other. Because the Web server has no information on whether there are consecutive requests from the same client, it cannot reuse the previous results only by basic functions, and the system has to perform the same task again. This increases the response time and degrades the usability.

To cope with this problem, Ruby on Rails provides session variables to store the information that exists when the connection is active. However, because the session variables are exchanged between the client and the server, a large amount of data cannot be stored into the session variables. Instead, the authors use the session variables to store the basic information, and the other variables used in the programs are stored in the database as objects and they are used for the subsequent requests. In general, an OODB (object-oriented database) is thought to be suitable for this purpose because the persistent object – an object stored into the database – can be treated in the same manner as the actual variable, while the programmers have to convert memory image of variables into database data format when they use an RDB. However, Ruby on Rails provides the ActiveRecord class. It abstracts the database functions such as SQL query and relations of tables and enables the programmers to consider the database objects as Ruby variables. Furthermore, Ruby on Rails also provides useful functions to maintain a relational database, such as migration and scaffold [5]. The authors adopted the RDB as the database; it also manages the user account and calculation results.

The relational database adopted in this system is SQLite [9]. Unlike other relational database, SQLite is incorporated into applications, while other database systems run as servers and provide client–server interfaces. The client–server-type database is suitable for large-scale applications. On the other hand, it is too large for a small application such as this system and requires more computer resources. Therefore, the authors


ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H)
FIT Code Parameter
get (application/pdf オブジェクト)
104%
検索
NBI 1
NBI 2
NBI 3
Birth Profile
PNBI
rho
PNBI[W/m^3]
Done

```
Deposition (1MW) 0.563[MW]
aperture corr. 100.58 %
birth(ap) 57.98 %
Deposition(ap) 0.566[MW]
Dep/Birth 97.10 %

Deposition (1MW) 0.522[MW]
aperture corr. 100.50 %
birth(ap) 58.66 %
Deposition(ap) 0.524[MW]
Dep/Birth 88.95 %

Deposition (1MW) 0.000[MW]
aperture corr. 0.00 %
birth(ap) 0.00 %
Deposition(ap) 0.000[MW]
Dep/Birth 0.00 %
```

Fig. 3. Results of the calculation.

selected SQLite as the database. The variables are converted into a binary stream using Ruby's marshalling mechanism and stored as BLOB (binary large object) data. The time when they are registered and referenced is recorded in the database, and they are used to dispose old objects from the database.

3.3. Supercomputer and analysis codes

The analysis codes are usually written in FORTRAN. However, the authors did not want to modify their source codes because they were developed by experts, and the authors were afraid if they would introduce bugs into the current system, which would be difficult to be found by the authors who are nonexperts. Therefore, the authors have treated the source codes as black boxes and followed their procedure to execute the programs. The supercomputer used in this system is NEC's SX-8. It is separated from other LANs due to security reasons. Therefore, it cannot be directly accessed from the other computers. The users have to log into the gateway servers that run on Linux or HP-UX to submit job requests to the supercomputer. The calculation is performed as batch programs. The batch queue is managed by NQSII, which is a batch system based on the PBS [10]. The job request is written in csh scripts, incorporating job management codes into it as comments. Here is an example of jobs script.

```
#PBS–S/bin/csh
#PBS–q A
cd/short/emo/src
setenv F_FF10 parameter.dat
setenv F_FF20 results.dat
./calc.go
```

The users can use SSH, Telnet, and FTP to log into the gateway servers. However, Telnet and FTP cannot be used outside the NIFS. The application server uses the SSH to connect to gateway servers to submit the jobs. The authors could use Telnet. However, because Telnet sends messages to remote servers without encrypting them, the user accounts and passwords might be eavesdropped upon when they are sent to the server. Unlike Telnet, SSH is an encrypted protocol that establishes secure communication. The other reason to choose SSH is that it can attain an exit status and can separately handle standard output and standard error. These functions are useful to invoke the programs remotely.

It takes several minutes to finish the calculation, depending on given parameters. Because the basic API of job management is not provided, the application server periodically checks the status of the job queue. When it detects the end of the queue, it retrieves the results using SFTP. The results are provided in text files and are converted into a graph (Fig. 3).

4. Discussion

As mentioned in the previous section, the cache mechanism is used to obtain a quick response. For example, in Fig. 2, when the user selects the shot number of the first row, the graph of the electron temperature distribution appears. If the data is found in the cache, the server reuses it without calculation. However, the average response time for the first call is 2.3 ± 0.4 (s) while that for the second call is 2.5 ± 1.3 (s). Because the first call does not use cache data, this result shows that the use of cache data degrades the response time. In this case, the calculation is relatively simple, and it does not take time. If the experimental data is large, the calculation time becomes longer, and the cache works effectively. This result suggests that the authors must tune the cache mechanism for providing a quick response.

5. Conclusion

To increase the usability of analysis codes running on supercomputers in batch mode, particularly for visiting scientists, the authors have developed a Web interface to the analysis codes in order to demonstrate the usefulness of this approach. In this system, the authors adopted Ruby on Rails as the main component. Because Ruby on Rails has base classes to provide the functions of AJAX and databases, programmers can efficiently develop Web-based applications.

Using this approach, the authors continue to develop Web interfaces for the existing analysis codes such as magnetic surface calculation programs. The magnetic surface information is one of the most basic parameters for analyzing plasma physics. However, the programs are not widely used in NIFS because they run on the supercomputers, and it is difficult to prepare input parameters in the required format. Therefore, many scientists request the assistance of other scientists to run the program for the required conditions. If the authors provide a user-friendly interface for the programs, the scientists will be able to conduct studies more efficiently.

Acknowledgements

The authors are grateful to all the researchers of the LHD project for their assistance and encouragement. The authors would also like to thank all the researchers in the NIFS for their assistance. This work was supported by the NIFS budget NIFS05ULHH512.

References

[1] S.P. Hirshman, W.I. van RIJ, Three-dimensional free boundary calculations using a spectral Green function method, Comp. Phys. Commun. 43 (1986) 143–155.
[2] K. Harafuji, T. Hayashi, T. Sato, Computational study of three-dimensional magnetohydrodynamic equilibria in toroidal helical systems, J. Comp. Phys. 81 (1989) 169–192.
[3] A. Fukuyama, M. Yagi, Burning plasma simulation initiative and its recent progress, J. Plasma Fusion 81 (2005) 747–754.
[4] http://www.adaptivepath.com/publications/essays/archives/000385.php.
[5] http://www.rubyonrails.org/.
[6] S. Murakami, U. Gasparinoa, H. Idei, S. Kubo, H. Maassberga, N. Marushchenkob, N. Nakajima, M. Romea, M. Okamoto, 5-D simulation study of suprathermal electron transport in non-axisymmetric plasmas, Nuc1. Fusion 40 (2000) 693–700.
[7] S. Murakami, N. Nakajima, M. Okamoto, Finite beta effects on the ICRF and NBI heating in the large helical device, Fusion Technol. 27 (Suppl. S) (1995) 256–259.
[8] http://www.fastcgi.com/.
[9] M. Owens, The Definitive Guide to SQLite, Apress, 2006.
[10] http://www.openpbs.org/.