Prototyping Efficient Desktop-as-a-Service for FPGA Based Cloud Computing Architecture

Shi Shu, Xiang Shen, Yongxin Zhu, Tian Huang, Shunqing Yan and Shiming Li
School of Microelectronics, Shanghai Jiao Tong University
Email: {shushi, shenxiang, zhuyongxin, huangtian, yanshunqing}@ic.sjtu.edu.cn
lishiming.cn@gmail.com

***Abstract*—Cloud computing, a delivery of computing as a service mainly implying how to use utilities in our context, can be provided either at infrastructure, platform or software levels. The Desktop-as-a-Service (DaaS) paradigm, derives from the software level Software-as-a-Service (SaaS) paradigm, is drawing increasing interest because of its transformation from desktops into a cost-efficient, scalable and comfortable subscription service. Unlike most existing solutions delivering service with various protocols based on image transmitting in PC dominating environment, we present a DaaS with cloud server technologies on FPGA to address the problem of high power consumption and heavy network traffic. With the booming of mobile cloud computing, users can access the service on demand with smart phones or other portable devices like iPad or Amazon kindle as well as PC. Our system provides virtual desktop web pages written in HTML/JavaScript to avoid frequent image transmissions and reduce network traffic. To build the cloud prototype system, we combine Lightweight TCP/IP stack (LwIP) and Java Optimized Processor (JOP) to build a web server enabling dynamic web page interactions. Our system significantly saves volumes of data in transmission and network bandwidth. Analytical performance evaluation shows that on average, our system suffers only 25% transmitting latency and saves 46% of energy efficiency in comparison to other solutions. Our efficient DaaS based on FPGA explores new application of embedded web server in green cloud computing as well as new service paradigm of mobile cloud computing.**

I. Introduction

Cloud computing architectures are rapidly spreading over the world of Information Technology (IT), supporting the idea of provisioning various computing capabilities "as-a-service", in a transparent way for users[1]. Among those various "services", DaaS, deriving from SaaS, has drawn considerable attention in recent years due to its formation of a significant interface over the gap between clustered servers and clients. Usually implemented with desktop virtualization technology, it has the potential to offer a new, cost-efficient, scalable and comfortable service on demand.

Cloud can have high utilization and thus great power efficiency. People are seeking for a green cloud computing architecture these years. Baliga et al. [2] compare the power consumption of different cloud paradigms and conventional computing. Berl et al. [3] review the usage of methods and technologies currently used for energy-efficient operation of computer hardware and network infrastructure. The paper also identifies some of the key challenges in green cloud computing. Liu et al. [4] present a new green cloud architecture to reduce power consumption of data center is presented. All these have shown the increasing concern of green cloud computing.

With the development of technology, DaaS's potential is drawing people's attention from personal computer into a thin client–the mobile platform, which has also been heavily impacted by cloud computing as well. Thousands of smart devices could access the internet, which will help drive the mobile cloud computing trend. In mobile systems, many devices have significant constraints imposed on them due to the requirement of smaller sizes, low power consumption and real-time issue.

DaaS paradigm complies with the client-server model that makes the information stored in servers on network and cached on clients. However, most implementations rely on continuous display synchronization between user interface on the client and application logic on the server through various protocols on data transmissions[5]. High network latency leads to high response times, which may exceed users' tolerance. High bandwidth results in frequent network upgrading and increases total energy costs. Furthermore, too many CPU computing resources that reserved for data processing and transmission lead to lower server efficiency and higher power consumption.

To address these problems, we propose a HW/SW co-design of an FPGA-based system to implement our DaaS infrastructure, which would produce several advantages: thin cloud ends, low communication bandwidth, high power efficiency and dynamic web enabled interaction. Our FPGA-based prototype provides virtual desktop web pages written in HTML/JavaScript with no special client software. We also merge JOP[6] and a web server to build DaaS server interacting with users' requests to cope with dynamic web pages, with which we could further implement more complicated applications.

The rest of the paper is organized as follows. Section II will introduce some related works about DaaS. We propose our design and implementation of our architecture in Section III. Section IV shows the experimental results. We conclude in Section V.

II. Related Work

Some versions of real-time remote desktop sharing software [7] like Netmeeting [8] and Remote Desktop bundled by Windows [9] operating system (OS) have been available for years. These allow users to exchange desktop screen images

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.121

between remote computers through the Internet. Netmeeting and Remote Desktop, however, force all users to install dedicated client software and use particular OS. Furthermore, the firewalls widely used by companies or organizations generally do not allow the connection of Netmeeting or Remote Desktop.

Virtual Network Computer (VNC) [10] is a remote desktop sharing software on various operating system. However, like Netmeeting and Remote Desktop, it is designed to support only port to port communications. As a result, VNC is not a good choice to present various services through different server clusters for better task management and better efficiency. In our expected system, users can access the same service within the same cluster while different services from different clusters. Like Netmeeting or Remote Desktop, VNC also uses a dedicated network protocol that differs from HTTP so that it is difficult for VNC connections to pass through corporate firewalls.

There are some web systems that support synchronous web browsing [11], [12]. Kobayashi [12], for instance, developed a web browser sharing tool for collaborative customer service. Ichimura and Matsushita[13], developed a QuickBoard web-server presentation system to help lecturers deliver their desktop screen images to student PC in class. However, they are designed only for dedication use.

Other WebOS-based general purpose solutions, including EyeOS[14] and Glide, can quickly build on-demand virtual desktop environment using web technology with low bandwidth cost and better performance.

Though our work resembles existing WebOS-based systems in terms of functions, our contribution is to achieve better power efficiency with reconfigurable devices, e.g. FPGA, as an enabling technology for green cloud computing. Although there also exist some on-chip web servers[15], their functions are limited and only support static web pages. In our implementation, we combine build a web server enabling dynamic web page interactions.

Other virtual desktop infrastructures, like Virtual Desktop Infrastructure (VDI), sometimes referred to virtual desktop interface is the server computing model combining server virtualization with remote presentation technology. It also suffers the same bandwidth and power issue as WebOS-based systems. All of these work indicates the demand of new efficient virtual desktop infrastructure.

III. Design and Implementation

Among the most related solutions, PC environment has always been the most popular platform. Though suffering limitations due to platform difference, we implement our solution on FPGA for better power efficiency and an exploration of FPGA as a generally-purpose device.

We developed a prototype web system using FPGA to store web pages, provide service and deal with requests. On architecture level, our system can be divided into two parts, the software parsing front-end and the hardware-based back-end.

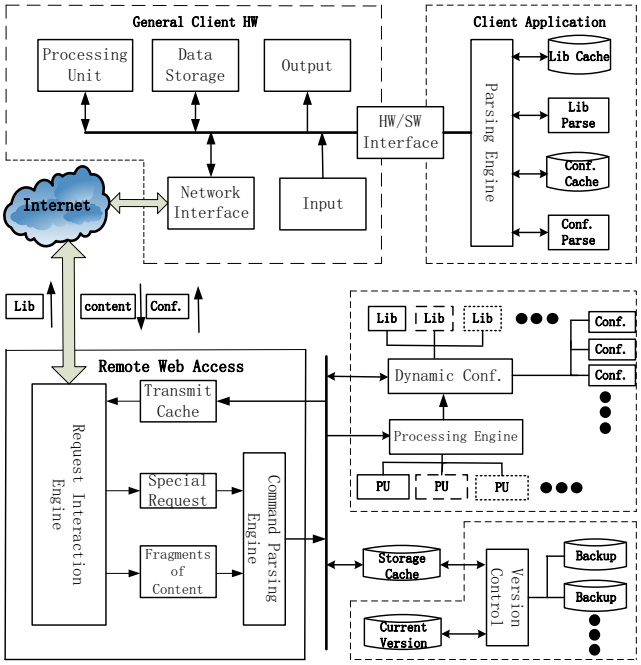

Fig. 1. System Architecture Prototype

A. Overall architecture

Fig.1 shows the system architecture of our prototyping system from a functional point of view. It consists of: (a) the general client hardware; (b) the client application; (c) the remote web access engine; (d) the processing engine; and (e)the data center. Internet, however, separates them into the so-called software parsing front-end in the top and the hardware-based back-end in the bottom. Despite all the actual implementation limits, we provide this robust system view with most common requirements considered, making provision for gradually porting onto FPGAs.

People who sign in for service should have devices providing similar functional structure with (a) and (b). Such devices need to do self-processing, access network, store some data and accept the users' input while present the output. Above such hardware layer, simple application should be adopted to cache remote libraries and configuration files while parsing them for output display and processing tasks.

On the server side, the Remote Web Access engine, takes the role of users' requests parsing, passing file content fragments to the data center while transferring special requests to certain commands for processing engine to cope with. Processing engine keeps real-time interactions with clients by transmitting library and configuration files. It also fetches commands to Processing Unit (PU) in which dynamic requests are handled, such as code compiling, scientific computing etc. Data center, on the other side, could be established on a separate cluster dealing with data backup and recovery.

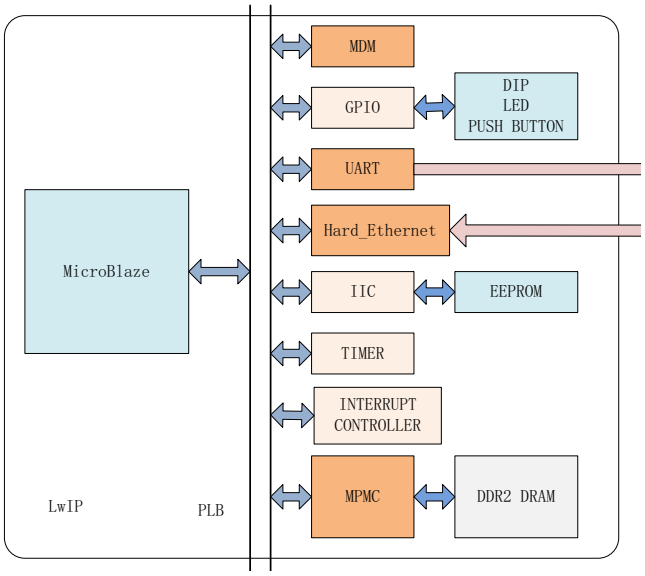

Fig. 2. Web Server Architecture Based on LwIP

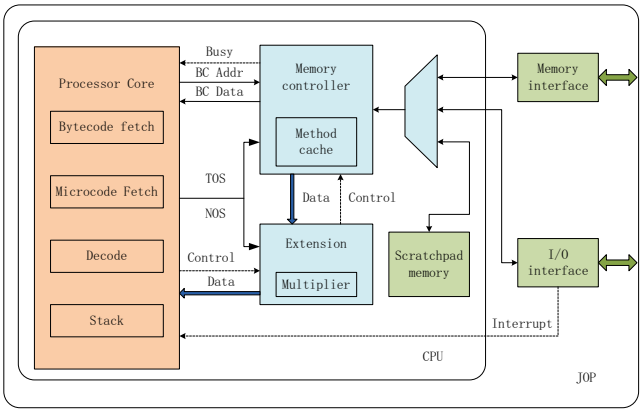

Fig. 3. Block diagram of JOP

B. Hardware platform partition

To estimate the feasibility of providing our services on FPGA as well as to pursue for less power consumption and better efficiency, we combine the TCP/IP stack LwIP and the hardware processing unit JOP together to accelerate our processing speed and form the hardware web server environment

1) Background:

a) Lightweight TCP/IP stack (LwIP): LwIP is an open source networking stack designed for embedded systems. It is provided under a BSD style license [16]. Xilinx provided adapters for Ethernet intellectual property (IP) core to add networking capability to an embedded system basing on LwIP library. As a result, web server can be shipped onto FPGA using LwIP.

LwIP works as software resource compiling into executable and linkable format and running on FPGAs. We implement the LwIP based web server as illustrated in Fig.2. The system integrated MicroBlaze, a Xilinx soft IP core processor, connecting to a Multi-Port Memory Controller (MPMC) with the other required peripherals using the Processor Local Bus (PLB) interface. MPMC, works as the interface for accessing memory while MicroBlaze Debug Module (MDM), interrupt controller, timer, Inter-Integrated Circuit (IIC), Universal Asynchronous Receiver/Transmitter (UART) are also those peripherals connected to MicroBlaze via PLB.

b) Java Optimized Processor (JOP): The Java optimized processor [6], [17] is an implementation of the Java Virtual Machine (JVM) in hardware. JOP translates the Java intermediate bytecodes to its own instruction set called microcode. These microcode instructions, implemented in hardware, are executed by the stack architecture. As shown in Fig.3, the CPU has a 4-stage pipeline, microcode fetch, decode and execute (stack) as well as an additional translation stage of bytecode fetch.

There is no direct connection between the processor core and the external world. The ports to the other modules are two elements of Top-Of-Stack (TOS) and Next-Of-Stack (NOS), input to the stack (Data), bytecode cache address and data, and some control signals [18].

The memory controller with method cache [19] implements the simple memory load and store operations and the field and array access bytecodes. The memory interface provides a connection between the main memory and the memory controller. The extension module controls data read and write. The busy signal is used to synchronize the processor core with the memory unit.

The I/O interface contains peripheral devices, such as the system time and timer interrupt for real-time thread scheduling, a serial interface and application-specific devices. Read and write through this module are controlled by the memory controller. All external devices are connected to the I/O interface.

2) Back-end system implementation: We designed a hardware system shown in Fig.4, called the micro-architecture.

A MicroBlaze soft-core processor works as the master device and shares memories with JOP. The shared memory, an actually dual-port block ram, is connected to JOP though Intellectual Property Interface (IPIF) [20]. With complete VHDL source and Xilinx Embedded Development Kit (EDK) as well as the help of IPIF, we wrapped JOP as a whole and connected it into our system.

MicroBlaze performs arithmetic and logic operations on data from memory and controls the working scheme of hardware modules via sending signals to PLB. On the right side of peripherals, two UARTs and one Ethernet are used to debug and display our results. One UART is the I/O interface of JOP and is used to download the jop bit stream that is compiled from java source file. It also displays the debug information about JOP. The other UART connects FPGA and the host computer is used to debug for the function of LwIP. The Ethernet IP core connects our system to the outside network so that users can access the system from client devices. MDM

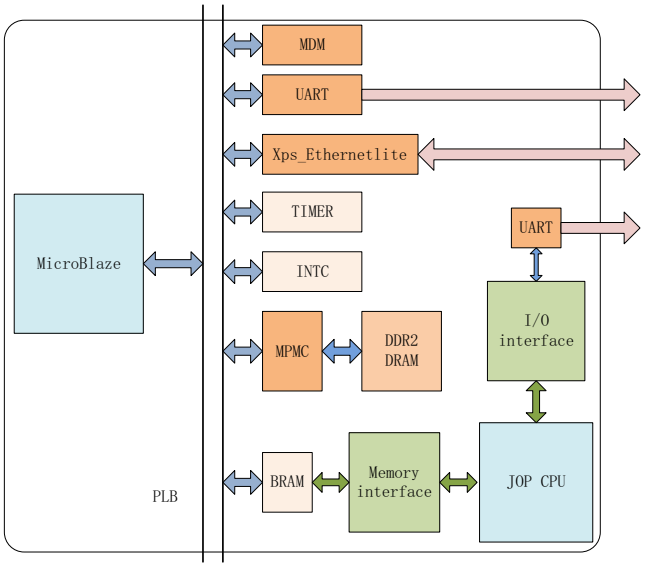

Fig. 4. Micro-architecture of FPGA-based Web-service module

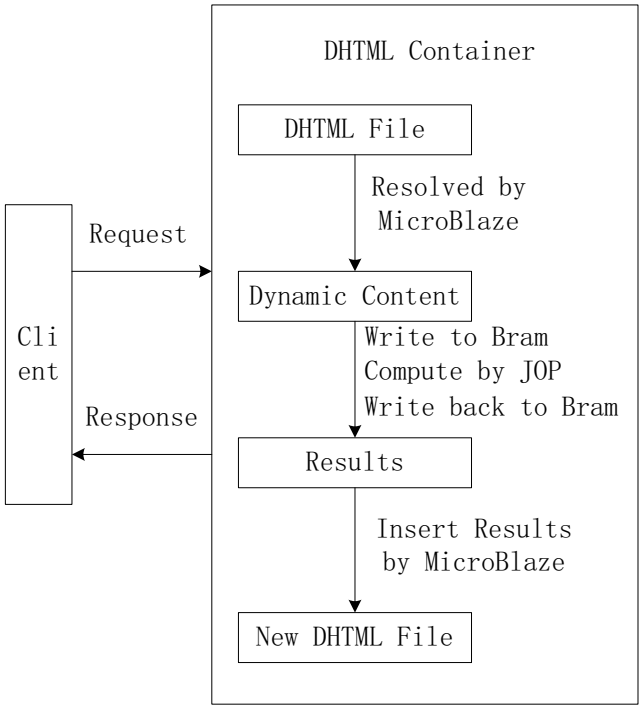

Fig. 5. Dynamic Requests Interaction Scheme

is used to download the web page file as well as the software application program. DRAM is used as file system memory storing web page file beginning at our predefined address of 0x92000000. For the remaining peripherals, timer and interrupt controller (INTC) are both essential for the control of MicroBlaze over the whole system.

3) Dynamic requests interaction scheme: Traditional FPGA-based web servers [21] only support static web pages as far as we know for they do not have a dynamic-web-page processing engine. However, static web pages have long been replaced by dynamic pages for lacking of capability on handling dynamically interactions. For some web services, like blog, wiki etc, most of their pages share similar page frames. If we use the static page scheme, pages will grow as the user number or word entry grows. On using dynamic scheme, however, many web pages can share the same frames. When sending response back to clients, the server could dynamically compose the page and the individual data together. As a result, we do not need such huge extra storage for similar pages.

Under our dynamic requests interaction scheme, after sending requests for dedicated pages by users, JOP dynamically rewrites the required pages with corresponding data inserted, and the LwIP-based web server sends the pages back to clients afterwards.

Fig.5 shows our dynamic requests interaction scheme. We call the hardware and software processing our web pages Dynamic Hypertext Markup Language (DHTML) Container which processes the request from client and sends response to the client. When the container receives the request from client, the DHTML file which is transported from client by TCP/IP protocol, is resolved by MicroBlaze. Then we can get the dynamic content that should be processed and stored. Next, the dynamic content is written to the share memory. We set up a flag in memory for Java program running on JOP and when the flag is enabled by MicroBlaze, the Java program begins to execute. Similarly, we set up another flag in memory for LwIP and the flag is enabled by Java program in JOP when the results are obtained and written back to bram in JOP. At this time, the right results are in share memory and can be visited by MicroBlaze. Then the results will be inserted into web page by MicroBlaze and new DHTML file (new web page) is generated. In the end, new web page is transported to client. One operation to dynamic web page is completed. This is the whole dynamic requests interaction scheme.

C. Software application partition

To make the system platform-independent, we should make the service web-accessible. Thus, we should focus on HTML and JavaScript which are composed into visible or audible web pages. HTML elements form the building blocks of all websites. It allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text. It also adopts Cascading Style Sheets (CSS) to define web page appearances and layouts while using JavaScript to affect web page behaviors.

Although web pages are stored on server side, they work mostly the same despite platform difference. We will discuss the web pages along with the front-end part as a whole.

1) Service organization: A typical view of a web page of virtual desktop is slightly similar to File Browser of Microsoft Windows. The web page is separated into three frames, header in the top, navigator in the left while contents of folders possessed most of the other space.

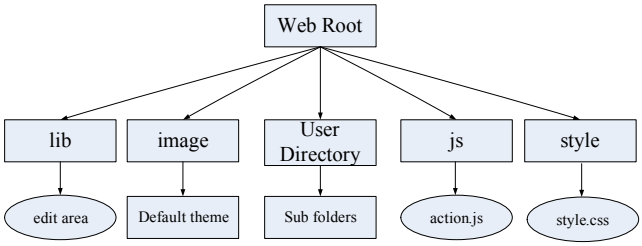

Fig. 6. Web root hierarchy

Fig.6 shows how we organized the whole web root hierarchy to present our services. The web root is the topmost directory where a web site is located that we can access the web server correspondingly with its domain name typed into the web browsers.

Folder “lib” contains 3rd-party open source libraries that can run on our server for dedicated use. For example, what is presented on the bottom left in Fig.7, “edit area” is a JavaScript editor library that enables us to input key words with syntax highlights when coding on distinct programming language source files.

Folder “image” stores the theme images for the virtual desktop system. As shown on the top left in Fig.7, it is a desktop theme we adopted which is similar to Mint-X-Metal from Mint Linux 10, a distribution derived from Ubuntu. Every folder or file will have a distinct file icon due to its Multipurpose Internet Mail Extensions (MIME) type.

Folder “js” controls animations on the web pages. It contains JavaScript files which could display image slides foreground while making background slightly opaque just as shown on the top right in Fig.7. Folder “style” consists of CSS files to affect the appearance and layout of the pages. Font type and color, background, image padding and margin.

The above folders do not need duplications for every single user because users share these libraries and configurations in common; actions and appearances they defined are almost same. User directories on the other hand, should be individually customized. Users store documents, pictures, videos etc. as they like. These personal folders compose the majority of the data storage. Since personal folders are independent from each other, we can divide these data into different server clusters.

2) Services implementation technologies: As a Desktop-as-a-Service, though no more than a prototype, we still need to realize some concrete services to prove its feasibility and advantages on FPGAs. For a common virtual desktop system, users are always willing to operate as if on their own computers. Be able to know what is in the current directory, to view documents and modify them are the basic requirements that users want to experience. For such reason, we implement several typical schemes to provide such requirements.

a) Directory scanning scheme: We do not adopt operating systems working on FPGA for lower resource utilization as well as for some necessity reasons. With an OS, unnecessary resource and power consumption would encumber the performance of the whole system. Our solitary goal, to some extent, is to bring out simpler and more lightweight implementation with the least performance and functionality loss.

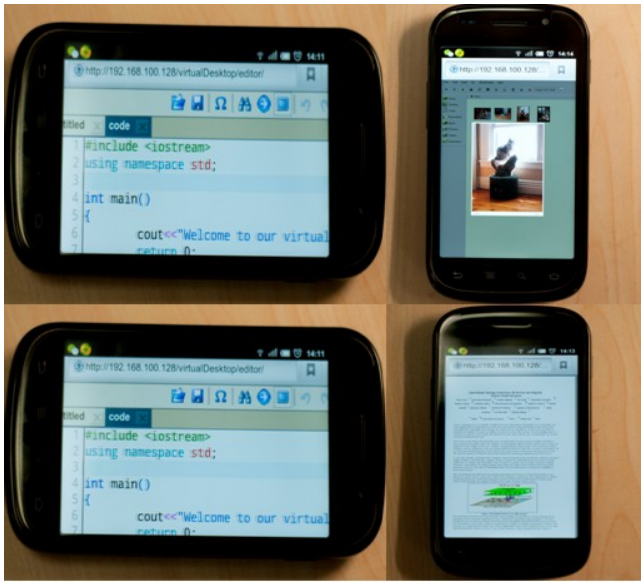

Fig. 7. Application services achieved for thin clients e.g. smart phones

Without the support of OS, we should take measures recording the current file directory. In our system, data files in JSON format are used as data records. With our solitary goal, JSON, or JavaScript Object Notation, a lightweight text-based open standard designed for human-readable data interchange, is used for serializing and transmitting structured data over a network connection. There is no need to have continuous data transmission between the server side and client side. Compared to XML, JSON is easier and more flexible. As a light weighted data interchange standard, JSON has such rich functionalities that it was even adopted by enterprise applications, e.g. IBM system directors. Although XML has some advantages such as high extension, for most web applications, there is no need to use complex XML to transmit data. Derived from JavaScript, JSON can be easily evaluated by JavaScript. The data structure of JSON is quite simple with name/value pairs and arrays. When users access their folder, the contents-recording JSON file will be transmitted to the client. Furthermore, when users take new file or delete operations, it will be treated as a post request sending to the web server. The server will dynamically modify the JSON file contents and send a feedback that makes the JavaScript to refresh the current web page on the client side. With information in JSON files transmitted to clients, browsers can support common user interface operations locally to reduce frequency of data interchanges between clients and servers, and achieve better efficiency at a system level.

b) Slides show scheme: Slides show is another typical application for users to enjoy viewing images. On every access, the web page examines the JSON file like the previous scheme illustrated. Then JavaScript dynamically rewrites the web page on the client side so that we can see all pictures

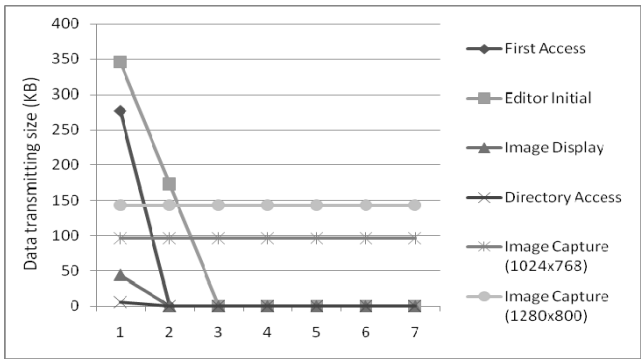

Fig. 8. Volumes of transmitted data for different requests (in KBs)

being well arranged.

On clicking the picture, animation starts that larger view of the picture will be presented in the center of the screen just like the right top in Fig.7 shows. What Fig.7 does not reveal clearly is that there are links named "prev" and "next" for users clicking to see the previous or the next picture. It is implemented with a combination of our own animation scripts and the open source JavaScript library called jQuery.

c) Documents viewer and editor scheme: There are hundreds of different documents formats in the world. Thanks to the dedicated file viewer applications, like Open Office, Adobe PDF readers or vi/emacs for instance, we can easily view or edit them on our personal computers. However, things got a bit tougher when we try to provide similar service without the OS support.

We edit different formats with different viewer application on our personal computers. Google Docs suggests that we had better keep all newly created documents in the same format that is defined by our own. When users want to keep distinct formats on clients, they can choose the type of pdf, doc or so to download after the format conversion has been done. In this case, we integrated an open source web editor named "edit area" to realize the word processing service. It is also syntax highlighted that we can even code on it. When we load or save files, asynchronously sending requests will get or save the file contents from/to the web server. These asynchronous requests use Asynchronous JavaScript and XML (AJAX) to send or retrieve data asynchronously from the server without interfering with the display and behavior of the existing page.

However, with those uploaded in various format, transforming may not be easy for an FPGA-based system. At this time, we use an affiliate PC environment to transform PDF or word docs into read-only html web page and put it back into the user folder. The bottom right of Fig.7 shows the effect of the web page that was transformed from the PDF files. File transformation is an algorithm work and it can be eventually written in programming language and ported into Java Optimized Processor hardware environment (later discussed) in our future work.

IV. Performance Evaluation

To measure the performance of our system, we firstly compare the total data size for each request on different platforms. Data size for most remote desktop sharing software depends on the screen resolution while our web-based system only relies on a small amount of web page sizes. Then we design a simple testing program to emulate browser user's operations, and use the program to measure the latency of basic operations, such as response to keystroke and image load. All the benchmarks are designed to be executed within a web browser to provide a common application environment across different platforms. We did not focus on the performance of web servers for the reason that LwIP is not comparable to the computer environment solutions like Apache or Internet Information Services (IIS). However, wrapped with JOP, LwIp has a performance gain over the original web server based on LwIP only. At last, we use i.e. Avalanche 2900 and Spirent TestCenter 3.51 to evaluate performance of apache web server and our system and then estimate their power efficiency accordingly.

A. Volumes of Data traffic

Image-transmitting-based sharing software is continuously sending compressed images to the client. As illustrated in Fig.8, for every transmitting interval, servers sent about 96 KB under the resolution of 1024 x 768, while 143 KB under that of 1280 x 800. In our system, however, users may download about 270 KB data for the first time to sign in, but will not suffer continuous data exchanging afterwards. This data includes some data for initialization that is not likely to change frequently, such as frame and basic data. With the data of the basic framework of desktop, when users access the system for the rest of times, they will not have to reload unmodified page resources due to the help of the browser cookie mechanism. If those data changes later, we just need to transmit the changed amount of data. From the second time on, sign in to the system will receive no more than 10 KB, just as little as accessing any directory shown in Fig.8. For other particular applications, file editing should download a 350 KB library file at first while image slides show should get a 50 KB preview.

Although our system needs to send relatively a bit larger volume of data in the beginning, it does not transmit data continuously. As a result, our system will not occupy much bandwidth.

B. UI operation latency

We measure both the latency and the data transferred for each of the four regular operation modes: key stroke, scroll the text, fill in the form and load bitmap images. These results are shown in Fig.9.

Here, we compare our system to two other platforms, the Microsoft Remote Desktop Protocol(RDP) windows and Citrix windows. Compared with other platforms, our system is able to perform those operations with little latency except for loading bitmap images due to the reason that our system is web-based and these operations are performed on the client

TABLE I
PERFORMANCE COMPARISON OF BLOCK IMAGE COMPRESSION AND DECOMPRESSION

	Max				Min				Norm			Time	
	ori	mat	jop	mb	ori	mat	jop	mb	mat	jop	mb	jop	mb
unit1	214	215	214	255	192	195	195	242	13.9642	11.2694	384.009	5299	18312
unit2	204	202	201	251	128	127	127	156	48.0000	45.6837	310.620	5300	18316
unit3	154	154	153	188	85	86	86	108	8.3066	5.4772	241.857	5300	18324
unit4	213	209	208	255	36	25	25	31	32.0312	30.3809	251.694	5299	18343
unit5	186	193	191	235	52	64	63	79	47.9375	45.5631	247.871	5298	18324
unit6	217	224	224	255	55	60	59	76	81.4862	79.4355	305.977	5301	18336
unit7	207	209	208	255	25	41	41	50	124.952	122.784	248.207	5298	18346
unit1	58	56	55	67	44	1954	44	57	8.8882	6.3246	89.2188	5293	18306

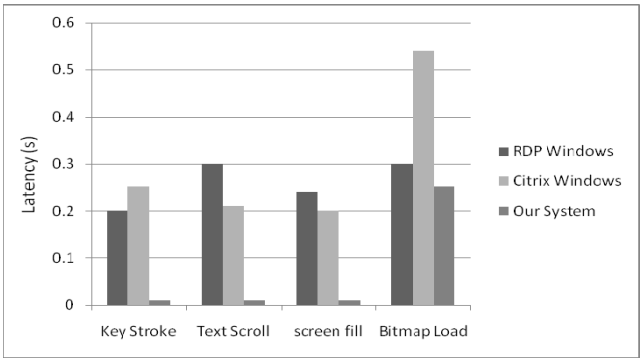

Fig. 9. Time latency of UI operations

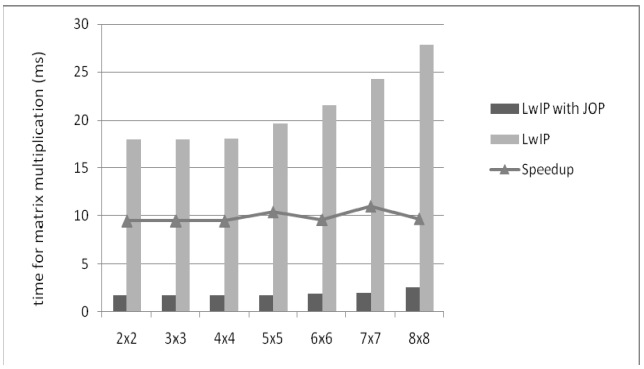

Fig. 10. Processing time of systems handling web pages containing matrix multiplications

side. When we need to interact with server, like downloading images, our system still does better than the other platforms. With relatively small data in transmission, our system does not suffer a long-time latency, which is only one fourth compared to other platforms.

C. Response time of CPU intensive tasks

Our web server is still not comparable to those software versions like Apache or nginx. This is due to the reason that LwIP runs inside a FPGA at 130-150Mbps to transmit/receive web traffics while Apache runs around 800Mbps [21].

However, combining JOP with LwIP in our system, some dedicated applications can be processed faster. This to some extent verifies the necessity of adopting specific accelerating

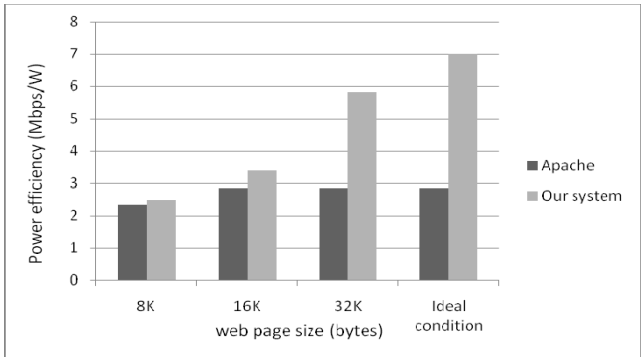

Fig. 11. Power efficiency of systems processing dynamic web pages containing matrix multiplications

modules in application fields of cloud system. As depicts in Fig.10, on processing matrix multiplication of different element sizes, LwIP with JOP costs around 2ms while LwIP version costs from 18ms to 27.8ms. The combination of JOP and LwIP gains an average 9.9 times speedup over that of LwIP-only version[16]. The minimum speedup is 9.5 while the maximum speedup is 11.0 and the average speedup is 9.9. Fortunately, the results can illustrate that our system is meaningful and obtains a relatively large speedup.

Also, the performance of block image compression and decompression in our system shows its advantage over that of LwIP-only version[16]. Table.I shows the results. We choose eight block images from the classical image Lena. "ori" represents the original pixel value. "mat" means the value recovered after matlab software. Max and Min part shows the maximal and minimal pixel value of block image respectively. "Norm" is used to estimate the degree of image distortion. The norm part of the table shows that the system with JOP has the best process ability, and it is much better than the system without JOP. The result executed by JOP is even better than that of matlab. The execution time by combining JOP with LwIP is almost 5300 clock ticks, which is one-third of that with LwIP-only version.

D. System power efficiency

We evaluated the Apache performance on a quad-core processor computer: Intel Xeon 5520. Results show that the throughput of Apache is about 600Mbps on transmitting small

pages and 800Mbps on transmitting larger pages. For our system, the number varies from 47 to 140Mbps at the average throughput on transmit and receive. On the other hand the power consumption of our FPGA system apache on CPU system is about 20w and 280w respectively.

We take throughput for every Watt as the power efficiency here to show how much performance could be generated with the same energy cost. As illustrated in Fig.11, without OS or peripherals, our system does 45.7% better than that of the comparing software system on average. If provided with a better-throughput-hardware to implement TCP/IP stack into hardware on a bigger FPGA, we believe the system could gain further improvement in the overall performance and energy efficiency. Also, our DaaS can be further implemented in pure hardware as a fixed part of IaaS.

V. Conclusions

In the hardware/software co-design of FPGA based DaaS prototype in the paper, we showed our efficient DaaS paradigm for green cloud computing. Due to the web-based software service implementation for thin clients, our system occupies much less bandwidth and takes one fourth of latency than those on other platforms. On the server side, we set up hardware server partition to solve dynamic web-page requests by integrating JOP with LwIP and achieved an average performance speedup of 9.9 times over the original LwIP TCP/IP stack. The power efficiency of our system is 45.7% better than that of Web service software running on generic CPUs on average, i.e. apache over Linux on CPU. Due to better throughput of hardware TCP/IP stack in our FPGA-based system, the overall performance and power efficiency can be further improved. The significant improvement in power efficiency indicates that reconfigurable hardware as an alternative to the current solutions and approaches to the cloud computing is promising. With our experimental results on FPGA and thin clients, we believe that the green cloud computing community will be convinced to convert more mature components of cloud computing into hardware-favored implementations to save precious energy.

The FPGA based DaaS prototype presented in the paper is a preliminary work to illustrate our efficient DaaS paradigm for green cloud computing. We would like to enrich our system functionalities as well as efficiencies. Functionality wise, our future work is to enable multiple tasks and users as our further proof of concept of DaaS paradigm. We plan to add more desktop applications to implement in our prototype system. To further improve system level efficiency, extensions to our current design include multiple JOP processors within one FPGA chip since multiprocessing design is an emerging trend for embedded systems. Those JOP processors will be also enriched with an embedded JAVA library. These work in our plan will make the efficient DaaS in our vision more practical with a better user experience.

Acknowledgment

This paper is partially sponsored by the National High-Technology Research and Development Program of China (863 Program) (No. 2009AA012201)

References

[1] S. Cristofaro, F. Bertini, D. D. Lamanna and R. Baldoni, Virtual Distro Dispatcher: a light-weight Desktop-as-a-Service solution. Proceedings of Cloud Computing Conference pp. 247-260, 2009

[2] J. Baliga, R. W. A. Ayre, K. Hinton and R. S. Tucker, Green cloud computing: Balancing energy in processing, storage, and transport, Proc. IEEE, 99(1), pp. 149-167, Jan. 2011

[3] A. Berl, E. Gelenbe, M. D. Girolamo, G. Giuliani, H. D. Meer, M. Q. Dang and K. Pentikousis, Energy-efficient cloud computing, The Computer Journal, 53(7), pp. 1045-1051, 2010

[4] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang and Y. Chen, GreenCloud: a new architecture for green data center, Proceedings of the 6th international conference industry, ACM, pp. 29-38, 2009

[5] X. F. Liao, H. Jin, L. T. Hu and X. J. Xiong, LVD: A light weighted virtual desktop management architecture, Proceedings of the 2nd International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and New Technologies (DMTF SVM 2008), Springer-Verlag, pp. 25-36, 2008

[6] M. Schoeberl, Jop: A java optimized processor for embedded real-time systems, Ph.D. dissertation, Vienna University of Technology, 2005

[7] M. Stefik, G. Foster, D. G. Bobrow., et al, Beyond the Chalkboard: Computer Supported for Collaboration and Problem Solving in Meetings, Communications of the ACM, 30(1), pp. 32-47, Jan. 1987

[8] R. Summers, Official Microsoft NetMeeting 2.1 Book, Microsoft Press.

[9] Microsoft Corporation, Netmeeting, http://www.microsoft.com/windows/netmeeting/

[10] J. R. Lange, P. A. Dinda and S. Rossoff, Experiences with client-based speculative remote display, Proceedings of USENIX 2008 Annual Technical Conference on Annual Technical Conference, pp. 419-432, 2008

[11] S. Greenberg, M. Roseman, GroupWeb: A WebBrowser as Real-Time Groupware, Proceedings of ACM SIGCHI '96 Conference Companion, pp. 271-272, 1996

[12] M. Kobayashi, M. Shinozaki, T. Sakairi, M. Touma, S. Daijavad and C. Wolf, Collaborative Customer Services Using Synchronous Web Browser Sharing, Proceedings of ACM CSCW'98, pp. 99-108, 1998

[13] S. Ichimura and Y. Matsushita, Lightweight desktop-sharing system for web browsers, Proceedings of the Third International Conference on Information Technology and Applications, pp. 136-141, 2005

[14] J. S. Pierce and J. Nichols, An infrastructure for extending applications' user experiences across multiple personal devices, Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology, pp. 101-110, 2008

[15] J. Riihijarvi, P. Mahonen, M. J. Saaranen, J. Roivainen and J. P. Soininen, Providing network connectivity for small appliances: a functionally minimized embedded Web server, IEEE Communication Magazine, 39(10), pp. 74-79, 2001

[16] S. MacMahon, N. Zang and A. Sarangi, LightWeight IP (lwIP) Application Examples, 2011, http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf.

[17] C. Pitter and M. Schoeberl, Performance evaluation of a Java chip-multiprocessor, Proceedings of the IEEE 3rd Symposium on Industrial Embedded Systems, pp. 34-42, 2008

[18] M. Schoeberl, JOP Reference Handbook: Building Embedded Systems with a Java Processor, ISBN:978-1438239699, 2009.

[19] M. Schoeberl, A time predictable instruction cache for a Java processor, Workshop on Java Technologies for Real-Time and Embedded Systems, pp. 371-382, 2004

[20] Xilinx, PLB IPIF (v2.02a), 2005, Xilinx Product specification, http://www.xilinx.com/support/documentation/ip_documentation/plb_ipif.pdf

[21] J. B. Yu, Y. X. Zhu, L. Xia, M. K. Qiu, Y. Z. Fu and G. G. Rong, Grounding High Efficiency Cloud Computing Architecture: HW-SW Co-Design and Implementation of a Stand-alone Web Server on FPGA, Proceedings of the Fourth International Conference on the Applications of Digital Information and Web Technologies, pp. 124-129, 2011