An Aspect Oriented Model for Software Energy Efficiency in Decentralised Servers

Samuel J. Chinenyeze, Xiaodong Liu and Ahmed Al-Dubai
School of Computing
Edinburgh Napier University
Edinburgh, United Kingdom
{s.chinenyeze, x.liu, a.al-dubai}@napier.ac.uk

***Abstract*—Green software is currently gaining interests with the increasing impact of IT in energy consumption. Green-ness in software however, can be achieved at various stages of the Software Development Life Cycle (SDLC). Consequently, several software engineering concepts can be adopted for achieving greener software. Aspect Oriented Programming (AOP) has been used in solving several crosscutting concerns of software, such as security and performance, but has not been well explored within the context of Energy Efficiency (EE). In this paper we propose and implement an Aspect-oriented Model for EE (AMEE) which adopts AOP for software EE as a crosscutting concern and consequently reducing computational energy consumption based on client-server architecture, where the server layer is distributed. By using a selected case study, the paper presents the energy saving outcome of using AMEE model for different simulated workload patterns.**

***Index Terms*—Green Software Engineering; Green Aspects; Greening Legacy Systems; Green Components; Aspect Oriented Programming.**

I. INTRODUCTION

Green software is a matter of concern with increasing energy bills and environmental impact of IT industry. As the environmental impact of IT can be directly observed through the computer hardware components [1], achieving greenness in software must involve a process which takes into account the underlying system/platform resources as shown in [2].

Furthermore, computer programs requires processing (or processor – CPU) time which consumes electric energy [1]. Since energy is saved when the CPU is in deep sleep state (idle time), a software system with low CPU utilization is considered more energy efficient (greener), if it completes a job nearly the same time as its counterpart [2], [3].

In this research we focus on reducing power consumption levels of the CPU – computational efficiency [2] by evaluating its utilization with respect to workload. Furthermore, task consolidation has always been limited to use of complex resource resolution structures in datacentres e.g. [4]. For our experiment we present that by employing the flexibility of Aspects at the server layer, multiple servers which service similar requests can collaborate to improve significant and overall energy efficiency either through task consolidation or resource sharing based on advices in AOP.

II. BACKGROUND INFORMATION

A. Green Software and Concerns

The term 'Green software' is commonly used to refer to software applications that efficiently monitors, manages and utilizes underlying resource(s) with little (or relatively minimal) negative impact on the environment [1]. Due to the high power consumption rates both in datacentres and other IT organizations, the green software research focus is mainly on energy efficiency [2]–[4].

Most green software techniques involve the use of software for monitoring and administrating power levels of hardware resources [5], as software has notable effect on power consumption from underlying hardware resources [1], [6]. Some issues leading to energy inefficiencies in software applications are improper analysis of requirements, suboptimal algorithms and inefficient resource allocation in applications [7]. Software applications however comprise different life-cycle phases [8]. The GREENSOFT Model [1] presents the environmental impacts of these phases (development, usage/runtime and end-of-life) of software life cycle while proposing tools and procedures to enhance software sustainability. In relating the software phases, Naumann, Dick et al. [1] further shows that environmental impacts of software is traceable to the development phase. Similarly, software runtime concerns can be addressed at the development phase [2], hence the concerns of the top level of abstraction (shown in Table I) is more specific to development.

TABLE I. SOFTWARE ENERGY CONCERNS

Levels of Abstraction	Concerns (runtime)	Agents
Level 1: Top	Suboptimal code and algorithms	Developer
Level 2: Middle	Mismanaged user input	User
Level 3: Bottom	High resource demand	Application-Platform

© 2014. The authors - Published by Atlantis Press

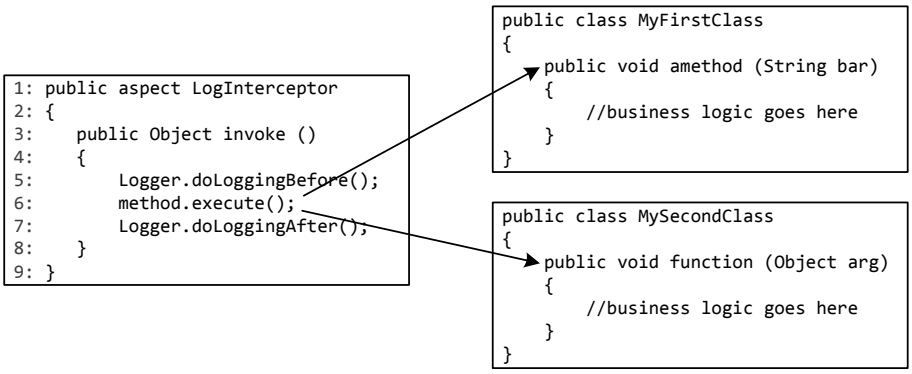

Fig. 1. Logging Example of Crosscutting in AOP (*snippets shown in Java*).

Furthermore, Kern, Dick, et al. [9] presents that users can influence software energy consumption at runtime through configurations. However, the notable cause of power consumption at runtime is due to resource overutilization [6].

The argument presented in Table I is that high resource demand as a runtime concern could be a consequence of mismanaged user input [9] or suboptimal algorithm [6], [7] which can be addressed at development phase [2] – hence, the levels of abstraction.

Furthermore, high resource demand of software at runtime is a platform concern which can be addressed by program instrumentation and monitoring [10]. Consequently, addressing overutilization of resources (layer 3 concern), can be fixed by manual source-code fixes and automatic code-optimization techniques [6] (by layer 1 agent). Several tools (and energy monitors) have been provided for instrumentation and monitoring energy consumption e.g. [5], [10], some of which provide Application Programming Interfaces (APIs) to allow flexibility of program control and per-process resource monitoring [11], [12].

B. *Why AOP?*

Aspect oriented programming (AOP) provides a component-based approach to the implementation of crosscutting concerns [13] as new requirements (functional or non-functional) of software systems. The redesign of software systems in most cases is not because they are functionally deficient – as their replacements are often functionally identical – but because they are difficult to maintain, port, or scale, or are too slow, or have been compromised by network hackers [8].

AOP and Crosscutting: AOP provides two types of crosscutting; dynamic crosscutting – which modifies the behavior of the program, and static crosscutting – which modifies the static structure of the types (classes, interfaces, and other aspects) and their weave-time behavior [13]. Dynamic crosscutting dominates the use of AOP [13], we therefore adopt this for our model.

The example of Figure 1 demonstrates how Aspects can be used (in a crosscutting manner) to alter the dynamic behaviour of a system (without directly modifying the original source code).

A benefit of using AOP technique and libraries (such as AspectJ [13]), is that crosscutting code can be implemented once as *aspects* (e.g. LogInterceptor of Figure 1). And within these aspects a developer can then define *where* to weave the code (e.g. Lines 5 and 7 of LogInterceptor) into *existing objects* (e.g. MyFirstClass and MySecondClass).

For clarification of AOP associated terms, some definitions have been given. Firstly, from Java context, AspectJ is the canonical Java library which implements AOP concepts. It adds to Java a few new constructs: pointcuts, advice, inter-type declarations and aspects. An *aspect* is an encapsulation of these new constructs and acts as the unit of modularity for crosscutting concerns, – analogical to Java classes, in behavior. An *advice* defines the code to execute upon reaching selected point(s) of execution. And a *pointcut* is a program construct that selects join points and collects join point context or data. In object-oriented programs join points consists of operations such as method calls, method executions, object instantiations, constructor executions, field references and handler executions [13][14]. Pointcuts and advice dynamically affect program flow [14] and will be adopted for our model implementation.

The Logger example in Figure 1 is one of many applications of AOP concepts in software engineering. Massive success has been achieved in the use of AOP to address security checks, performance, transaction management etc. [13] as crosscutting concerns, most of which are non-functional requirements (such as performance and security). Consequently, to achieve energy efficiency as a non-functional requirement for green software inspires our idea of employing AOP techniques such as dynamic crosscutting.

III. Related Work

Although Energy Efficiency (EE) has been a growing issue in IT, the issue of applicability to software developers is also an area of interest. Consequently, resource monitoring and software energy metering are critical in achieving energy efficiency in software [2]. Some work has been done to address energy-efficiency of distributed systems based on resource monitoring and algorithms, however focused solely

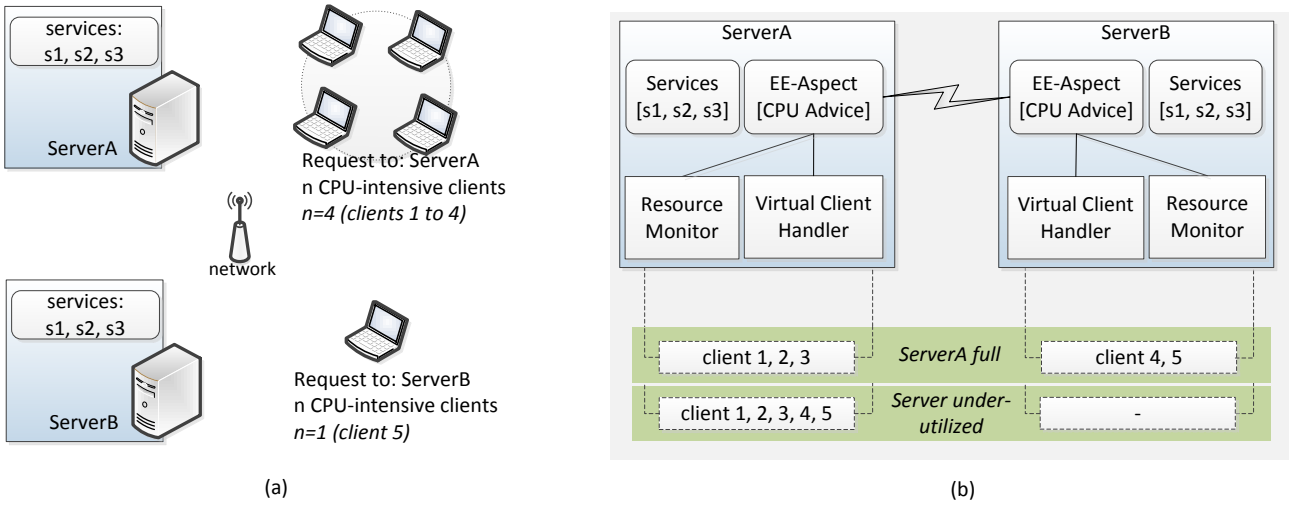

Fig. 2. The (a) Scenario and (b) AMEE Model.

on datacentres [4]. [3] closely considers the software development phase by establishing that architectures has effect on software power consumption given the CPU specification, however choice of software architectures are considerable for design phase before implementation [8] which raises a question of re-inventing the wheel for greening legacy systems. We fill this gap by considering EE as a crosscutting concern. Also, the load-balancing feature which our algorithm implements, can be achieved by traditional software load balancers e.g. vanish [15], although EE is not the main aim but could be a consequence of utilizing traditional software load balancers. However, we focus on reducing energy consumption as a crosscutting concern. We take into consideration, resource monitoring and energy metering for distributed systems using AOP technique. Several works present use of AOP for addressing crosscutting concerns such as performance, security etc.[13], similarly we demonstrate that AOP can be employed for software EE using dynamic crosscutting in AOP to modify system behaviours based on the current system state or predefined condition such as threshold – as discussed in the next section.

IV. The Model

Energy efficiency in software systems is often viewed in terms of power management. Power management is an aspect of optimisation that can be achieved by any of the three approaches [16]; implementing the system to

1. do less work or balancing of load for minimal execution [15],
2. consolidate processes to eliminate resource overutilization [2], [4], and/or
3. turn off idle elements [4].

Our Aspect-driven model (Figure 2) focuses on power management by an efficient combination of points 1 and 2, in a case of server under-utilization or overload. The model is built on client-server architecture, where the server layer is a collection of pervasive entities which can interact to efficiently handle requests. Request handling at the server layer is controlled by the relative resource consumption state of the server's resources.

A. Resource Monitor

The resource monitor uses a system monitoring API (such as SIGAR [12]) to provide resource consumption levels such as percentage CPU utilization which are utilized by the EE-Aspect. The resource monitor is implemented as a thread which runs independent of the services provided by the server.

B. EE-Aspect

In an AOP context, an advice defines the code to execute upon reaching selected point(s) of execution [13]. We use the term "EE-Aspect" to refer to the AOP module which defines an advice (and algorithm) for the resource of concern (such as CPU advice) – see *Overload Scenario Algorithm* section. Particularly, our use of "EE-Aspect" does not imply that AOP alone can achieve EE, but rather signifies an Aspect which treats EE as a crosscutting concern – aimed at reducing energy consumption per task. Furthermore, system resource levels are shared by the server using the EE-Aspect.

C. Virtual Client Handler (VCH)

By utilizing the information from the system monitor, the Aspect frequent-checks redirects client task (with the Virtual Client Handler) to alternate server if current server is full, or consolidates task from a server to another utilized server if servers are under-utilized. The VCH also keeps a count of redirected requests.

D. Overload Scenario Algorithm

The Scenario (Greening Legacy System): Clients 1, 2, 3 and 4 requests for service from ServerA and Client 5 from

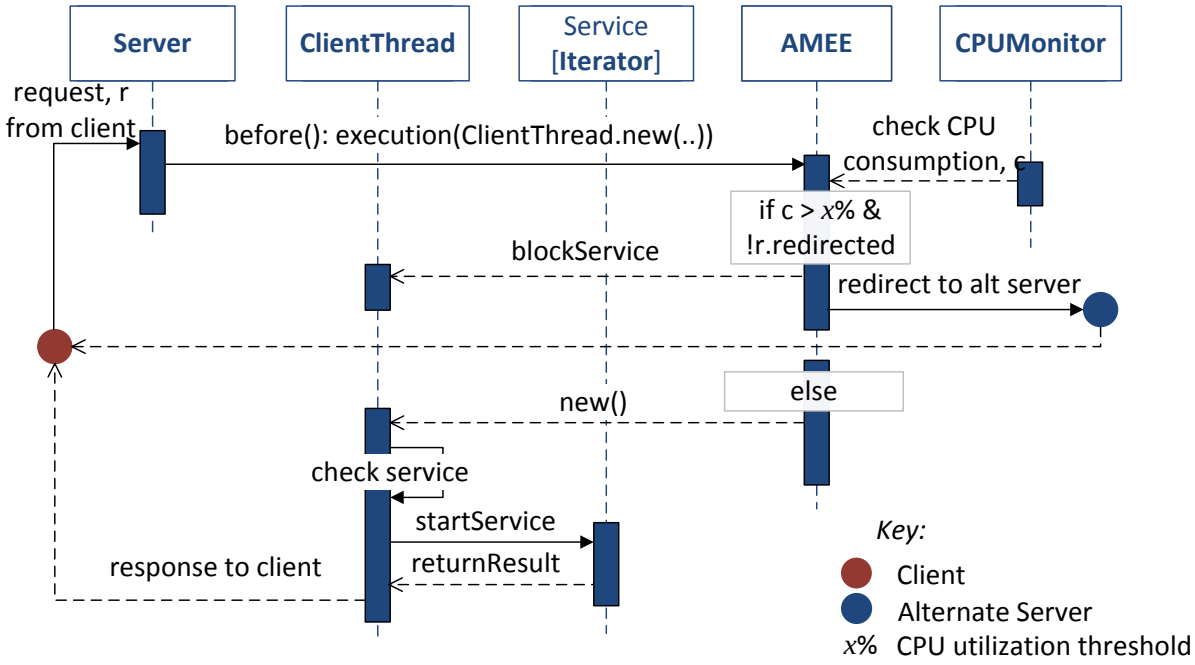

Fig. 3. Sequence Diagram for System Implementation.

```
// Aspect: EE-Aspect on main_server

1: Continuous_check: monitor cpu_consumption
2: Pointcut: before handling new client_request
3:
4: Advice:
5: IF (cpu_consumption > threshold)
6:  {
7:    block service on main_server;
8:    VCH: connect to free_server;
9:    VCH: redirect client_request to free_server;
10:   VCH: retrieve client_response from free_server;
11:   send response from free_server to client;
12: }
```

Aspect 1. Algorithm for Overload scenario

ServerB. In the traditional client-server system, servers handle requests only from directly connected clients. However, for EE, servers collaborate and service requests based on overall system consumption rate (i.e. for two possible scenarios: overutilization/overload or underutilization of resources).

The overload scenario algorithm (see Aspect 1) is wrapped-up in an AOP advice, as shown in lines 4-12 (implemented in the EE-Aspect component). The pointcut ensures that the check in the algorithm is run prior to handling client requests on the server.

The advice runs a check using *cpu consumption state* (from the resource monitoring API) and *a set threshold* (which is the choice maximum % utilization) to manage/control the resource usage at the server layer and consequently reduce the system's power consumption per task.

V. Experiments

A. System Implementation

We implement the system and Aspect Model for EE (AMEE) using Java with Eclipse IDE on a client-server based architecture. For support of multi-clients at the server, we make use of Java threads to concurrently handle client requests. Furthermore, SIGAR API for Java [12] (System Information Gatherer And Reporter) was employed for CPU monitoring, with AspectJ plugin for Eclipse as our aspect-oriented Java extension for implementing the CPU advice.

The Client application was implemented as a simple Java client (single class), which takes three arguments (server IP address, port and service request) for making the connection to the server. The Implementation (see Figure 3) shows three different primary entities; client, main server and alternate server. The main server and alternate server are of same class composition. Figure 3 presents sequence for the main server; that is, the server which a client request is initially directed to.

If CPU consumption exceeds the set threshold (in Figure 3 as c > x%), the Aspect redirects the request to the alternate server (with low CPU utilization). An extra check is added (in Figure 3 as !r.redirected) to avoid redirecting a request that was already redirected.

We obtain the time length of each client's request to the server by the difference in the send and receive time at the client (using Java timestamp facility similar to [3]). Subsequently we present the overall system performance as

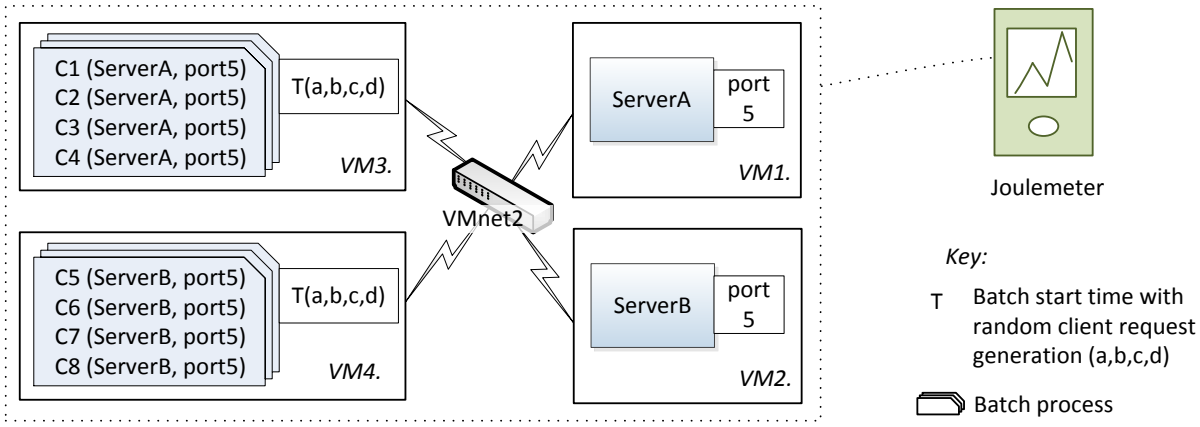

Fig. 4. Environment Setup for Experiment.

the difference in the start/request time of the first client and the finish/response time of the last client.

The Service package: To simulate a CPU intensive task we implemented an Iterator class as the service provided by the Server. The Iterator class performs string manipulation with for-loop for 50,000,000 times and returns the length of string when loop is completed. To reduce threats to validity (due to string manipulations) we randomly generate the client requests (through batches as shown in next section) and further repeat the experiment.

B. Deployment Settings

Figure 4 presents the environment setup for the deployment of our experiment. To run the experiment, a Windows 7 Virtual Machine (VM) was set up using VMware Workstation and the following configurations; 1GB Memory, Intel Core i7 2.20GHz CPU (on VM:1 Processor 2 Cores), and 20GB Hard Disk. We then cloned (full clone) the VM into three VMs.

Subsequently, two different servers, Server A and Server B are setup on two different VMs, VM1 and VM2 respectively – with the servers listening on port 5 (as shown in Figure 4).

For random request generation (multi-client simulation), we build a Java batch program to randomly start up four clients at different VMs (VM3 and VM4). The VM3 and VM4 batch begin execution at the same time. The batch process randomly fires up a client request (at random times; Ta, Tb, Tc, Td for each client, determined by Java random number generator API) until four clients are reached on respective VMs.

Furthermore, for VM power metering we use Joulemeter [10], and we employ experimental repetition to minimize inaccuracies in Joulemeter VM power estimation presented in [10]. The same deployment configuration is used to deploy two systems: one with our Aspect model for EE and another without. Note that; Joulemeter gives the CPU power consumption (in W) for every second (s) timestamp (as shown in later discussed Figures 4 and 5). However, we calculate the energy usage to complete given tasks in Joules (Ws) using the provided Joulemeter readings.

Set Parameters: The experiment involving our model was simulated using 65% CPU utilization threshold. This threshold is used since the clients involved are fewer in number and requests are run concurrently. Different factors can influence the choice of threshold, including; the type of

TABLE II. RESULTS OF EXPERIMENT

Client	Set 1: Time (ms[ss])		Set 2: Time (ms[ss])		Set 3: Time (ms[ss])	
	With AMEE	***No AMEE***	***With AMEE***	***No AMEE***	***With AMEE***	***No AMEE***
1	25368[01]	33998[01]	31765[01]	36756[01]	24562[01]	23752[01]
2	27799[04]	39065[03]	45679[02]R	38721[02]	27256[07]	32809[07]
3	33327[14]R	39408[04]	31061[02]	39298[02]	39969[08]R	33729[08]
4	29844[18]R	38470[05]	44883[03]R	38674[03]	38923[09]R	32824[09]
5	36634[02]	25646[02]	45804[02]	41841[02]	41160[01]	34576[01]
6	37101[03]	31243[05]	47675[02]	41077[02]	42953[02]	36245[02]
7	27489[05]R	30314[15]	46927[02]	40530[02]	42577[03]	36478[03]
8	37409[06]R	28861[17]	32387[03]R	39968[03]	27535[07]R	34118[07]
Time Total	45s	42s	47s	41s	44s	39s
Consumption	396.6J	442.2J	410.8J	429.4J	396.2J	419.2J

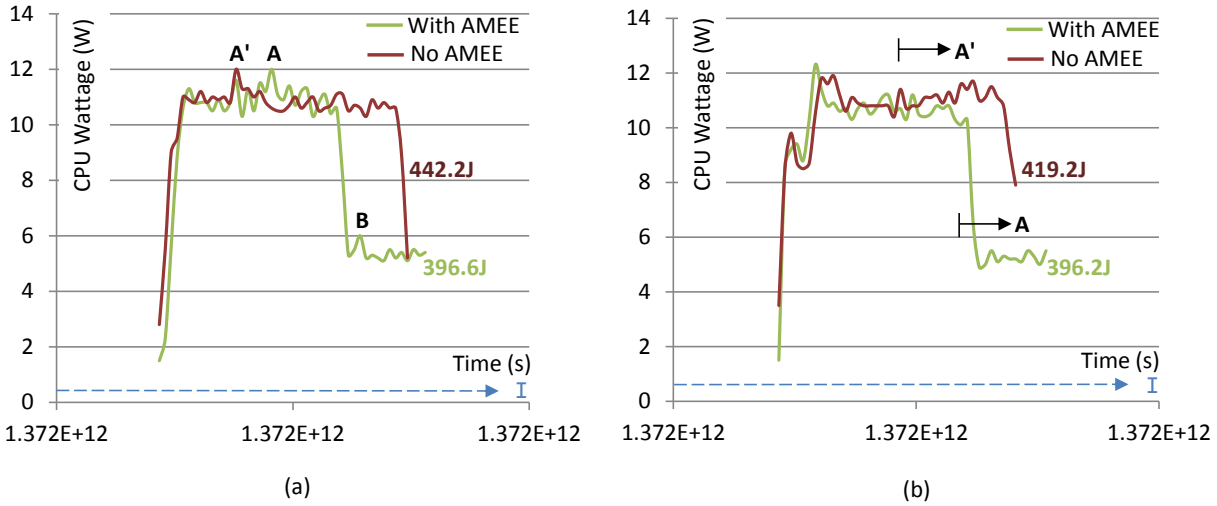

Fig. 5. (a) Set 1 and (b) Set 3 Analysis: Dispersed Requests.

application (single or multi-threaded), the demand/workload on the application (how high?), the type of and amount of CPU cores of the host system, etc. The case would be that, by using our Aspect Model for EE (AMEE) with random client request generation, clients would be redirected to the alternate server if the main server is full.

C. Results and Discussion

The experiment is conducted for two scenarios using same deployment settings:

1. Server layer with our proposed Aspect Model for EE (*With AMEE*) and
2. Non-EE optimised server layer (*No AMEE*).

Key for Table II: The experimental results are presented in **Time** (ms[ss]) column. The time column shows the response time in milliseconds, and in square brackets; the start-time (obtained from the timestamp seconds unit, i.e. ss in hh:mm:ss) for each randomly generated client. **R** signifies that the client concerned was redirected to an alternate server.

As shown in Table II, we present three results of the experiment based on randomly generated client requests to depict different workload scenarios of operation – we discuss these scenarios in this section with their associated graphs. The aim of the workload scenarios is to show the effect of the distribution of a fixed number of client requests on the server in relation to the AMEE proposed model.

Legend for Figure 5 and 6: The features with a prime sign (e.g. A') are used for No AMEE scenario while features without prime sign (e.g. A) are used for With AMEE scenario. Also, the feature, I, denotes the average idle consumption which is about 0.2W.

- ***Dispersed Requests.*** The first and third result set (also shown in Figure 5) are classified as a scenario where requests to the server are dispersed. In such dispersed scenario there is more energy savings by adopting AMEE.

Different features are presented in the Figure 5. The feature A', in the non-optimised scenario of Set 1 is a peak derived from sending/starting closely packed requests (from clients 3, 4 and 6) to the servers, and with the dispersed time difference in the requests the wattage is stable with small triggered peaks but still at the top of the graph (as seen in A' of Set 3), as opposed to A of Set 3 (also seen in B of Set 1) which falls due to redirect. The feature, A and B of Set 1 are the peak power consumption from use of AMEE traced to the redirection of client requests, a consequence of registering the clients to an alternate server (similar to the mobile experiment of [17]).

Due to the dispersion in request time (shown in [ss] column), the allocation of requests to server (by AMEE – using the overload algorithm) saves more energy (about 45J in Set 1 and 23J in Set 3). However, since the AMEE and non-AMEE scenario for Set 3 were produced with the same timing in client request generation, we prioritise the energy savings for Set 3 dispersed scenario which is 5.49% savings.

The energy savings in Set 1 can be further interpreted as resulting from: client 7 and 8 of (AMEE scenario) being consolidated to server A while client 3 and 4 requests were redirected to server B since the CPU resource of server A was overloaded. Furthermore, the cost of starting more cores for client 7 and 8 at the time (ss: 15 and 17) resulted in more computational power at the no-AMEE scenario of Set 1.

- ***Clustered Requests***. The AMEE and non-AMEE scenario for Set 2 were produced with the same timing in client request generation. Set 2 shows the use of AMEE to achieve an energy saving of 4.33%. The workload pattern and the CPU energy consumption for Set 2 are presented in Figure 6. From Figure 6, the feature A', is the peak CPU power consumption observed with the no AMEE scenario (i.e. prior to optimization). The feature A, however shows a parallel reduction in the peak wattage – this was traced from the redirection of client 2 request (see Table II). We therefore

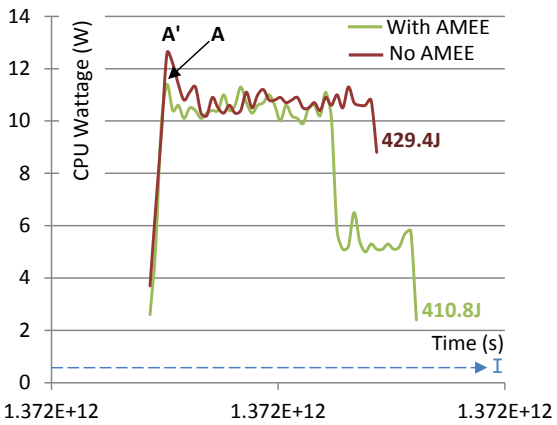

Fig. 6. Set 2 Analysis: Clustered Requests.

deduce that, optimization attempts that aim to lower different peaks in wattage can achieve overall system energy savings, for any given time interval.

At all scenarios of AMEE result samples, however, there seem to be an overall delay in response due to the redirection of requests, the reason being due to our focus on reduction of energy consumption through our model. With the focus on reduction of energy consumption, AMEE control mechanism was based on resource consumption, as shown in the algorithm (see Aspect 1 presented earlier):

```
Advice:
IF (cpu_consumption > threshold) THEN (ACTION)
```

To extend efficiency an overall monitoring of other software quality attribute has to be taken into consideration at the EE-Aspect component of the AMEE model. For instance: a balance between performance and energy consumption may implement an algorithm as follows:

```
Advice:
IF (cpu_consumption > threshold) AND
(cpu_performance is pi) THEN (ACTION)
```

As presented above; cpu performance is a factor of cpu specification/speed, and client pool size at the VCH in AMEE – which is implementable in ENUMs – denoted by P_i states, e.g.:

```
p1 = {cpu_type_1, pool_size_1},
p2 = {cpu_type_2, pool_size_2},
…
```

The recommended algorithm above would consequently resolve the delay in responses to client requests – caused by redirection, R.

Conclusively, the results show that by using AMEE and understanding workload patterns we can save energy. The experiment in this paper proved that AOP can be employed in server systems with CPU-intensive services to monitor and reduce over-all system energy consumption caused by software processes, thus addressing the layer 3 concern (high resource demand) mentioned in section 2. AMEE manages the distribution of requests, and to reduce performance overhead redirection is done once. This is so because unmanaged distribution of requests is a source of long resource consumption and increased waiting time for a shared resource [3].

VI. Conclusions and Future Work

This paper has been able to present a new model which adopts AOP to address the issue of software energy consumption, by treating EE as a crosscutting concern. The Aspect-Oriented Model proposed was defined to encapsulate all the 'EE concerns' of a system (CPU energy consumption in our case study). The consequence of the encapsulation of EE concerns achieved reusability of the EE component module across multi-server systems.

The AMEE model can be used to complement existing software architecture design process to improve runtime energy concerns. It could be argued that re-directing a client request to different server for reduction of energy consumption per task could result in a performance overhead; the criteria for re-direction however, can be further optimised based on different considerations such as work load patterns.

Since our proposal was based on virtualised environments, future work will include evaluation of the model using power meters and physical machines. Also we focused the current work on AMEE model on CPU resource, for future work we look to expand the model to take into consideration other system resources, which raise EE concerns like disk, network and memory.

References

[1] S. Naumann, M. Dick, E. Kern, and T. Johann, "The GREENSOFT Model: A reference model for green and sustainable software and its engineering," *Sustain. Comput. Informatics Syst.*, vol. 1, no. 4, pp. 294–304, Dec. 2011.

[2] B. Steigerwald and A. Agrawal, "Developing Green Software | Intel® Developer Zone," 2011. [Online]. Available: http://software.intel.com/en-us/articles/developing-green-software. [Accessed: 16-May-2014].

[3] B. Zhong, M. F. M. Feng, and C.-H. L. C.-H. Lung, "A Green Computing Based Architecture Comparison and Analysis," *2010 IEEEACM Intl Conf. Green Comput. Commun. Intl Conf. Cyber Phys. Soc. Comput.*, pp. 386–391, Dec. 2010.

[4] C.-H. Hsu, S.-C. Chen, C.-C. Lee, H.-Y. Chang, K.-C. Lai, K.-C. Li, and C. Rong, "Energy-Aware Task Consolidation Technique for Cloud Computing," in *2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, 2011, pp. 115–121.

[5] N. Amsel and B. Tomlinson, "Green Tracker : A Tool for Estimating the Energy Consumption of Software," in *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, 2010, pp. 3337–3342.

[6] G. Software, “Software Bloat and Wasted Joules : Is Modularity a Hurdle to Green Software?,” no. September, pp. 97–101, 2011.

[7] D. Rogers and U. Homann, “Application Patterns for Green IT,” *The Architecture Journal - Green Computing*, 2009. [Online]. Available: http://msdn.microsoft.com/en-us/architecture/dd393307. [Accessed: 16-May-2014].

[8] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Boston: Addison-Wesley Professional, 2003.

[9] E. Kern, M. Dick, T. Johann, and S. Naumann, “Green Software and Green IT: An End Users Perspective,” vol. 3, pp. 199–211, 2011.

[10] A. Kansal, F. Zhao, and A. A. Bhattacharya, “Virtual Machine Power Metering and Provisioning,” pp. 39–50.

[11] I. Ari and N. Muhtaroglu, “Design and implementation of a cloud computing service for finite element analysis,” *Adv. Eng. Softw.*, vol. 60–61, pp. 122–135, Jun. 2013.

[12] R. Morgan and D. MacEachern, “SIGAR - System Information Gatherer And Reporter,” 2010. [Online]. Available: https://support.hyperic.com/display/SIGAR/Home. [Accessed: 16-May-2014].

[13] R. Laddad, *AspectJ In Action: Enterprise AOP with Spring Applications*, Second Ed. Manning Publications Co., 2010.

[14] “The AspectJ (TM) Programming Guide.” [Online]. Available: http://www.eclipse.org/aspectj/doc/released/progguide/index.html. [Accessed: 16-May-2014].

[15] Varnish Software, “Load balancing with Varnish.” [Online]. Available: https://www.varnish-cache.org/trac/wiki/LoadBalancing. [Accessed: 16-May-2014].

[16] W. Vereecken, W. Van Heddeghem, D. Colle, M. Pickavet, and P. Demeester, “Overall ICT footprint and green communication technologies,” *2010 4th Int. Symp. Commun. Control Signal Process.*, pp. 1–6, Mar. 2010.

[17] C. Siebra, P. Costa, R. Miranda, F. Q. B. Silva, and A. Santos, “The software perspective for energy-efficient mobile applications development,” *Proc. 10th Int. Conf. Adv. Mob. Comput. Multimed. - MoMM ’12*, p. 143, 2012.