ISSN 2278-3091
Volume 9, No.5, September - October 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse163952020.pdf
https://doi.org/10.30534/ijatcse/2020/163952020

Energy Efficient Software Development Techniques for Cloud based Applications

Aeshah A. Alsayyah[1], Shakeel Ahmed[2]
[1,2] College of Computer Sciences and Information Technology, King Faisal University, Al-Ahsa, Saudi Arabia
aishamonem@gmail.com, shakeel@kfu.edu.sa

ABSTRACT

Worldwide organizations use the benefits offered by Cloud Computing (CC) to store data, software and programs. While running hugely complicated and sophisticated software on cloud requires more energy that causes global warming and affects environment. Most of the time energy consumption is wasted and it is required to explore opportunities to reduce emission of carbon in CC environment to save energy. Many improvements can be done in regard to energy efficiency from the software perspective by considering and paying attention on the energy consumption aspects of software's that run on cloud infrastructure. The aim of the current research is to propose a framework with an additional phase called parameterized development phase to be incorporated along with the traditional Software Development Life cycle (SDLC) where the developers need to consider the suggested techniques during software implementation to utilize low energy for running software on the cloud and contribute in green computing. Experiments have been carried out and the results prove that the suggested techniques and methods has enabled in achieving energy consumption.

Key words: Cloud Computing, Energy-efficient, Green computing, Software Development Life cycle,

1. INTRODUCTION

The usage of green cloud computing became widely thought of issue in large organizations. The large organizations have targets to minimize the energy consumption and reduce unnecessary resources. Cloud computing can help effectively in this situation by allowing different services include storage with different capacity, data processing, sharing information and so on [1]. This improvements related to hardware side by migrating the software, data and most computing to CC. whilst, achieve great deal of efficiency by having hardware that required minimal energy. Cloud computing is cost effective and highly scalable infrastructure for running High Performance Computing (HPC), enterprise and web applications. Moreover, there has been drastically growing demand of cloud infrastructure by different organizations which has resulted in the increased consumption of energy to run these data centers, which has become a critical issue. Where by High energy consumption increases the high operational cost and reduces the profit margin of cloud providers. Further, this leads to high carbon emissions which is not eco-friendly [2].

Further, there is a need to consider the issues related to energy consumption of cloud environment. So, the applications or software must be prepared with energy efficient techniques at the implementation phase. This means that the developers need to design the software with Green Software Development Life Cycle (G-SDLC). Parameterized phase will support and enhance the source code of software; by identifying and analyzing the hotspots in code, that causes energy consumption with help of tools or methods. Cloud computing and software development are the backbone to design software which is most energy efficient for cloud based applications. They can be merged as one framework of green cloud computing [3]. The framework shown in Figure 1, contains two major components: software and cloud structure.

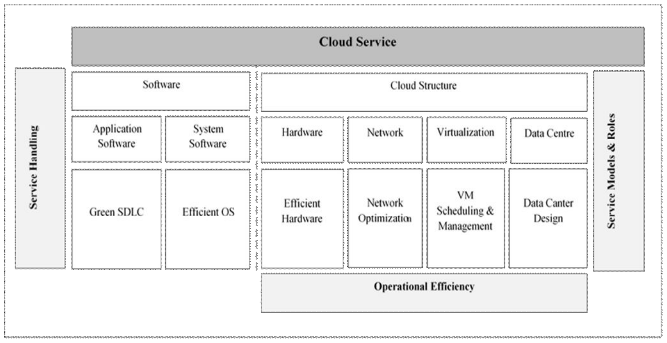


Figure 1: Framework of green cloud computing [3]

Complex software application can deliver high efficient, accurate and powerful work to modern organizations. However, it causes resource and power consumption that will lead to a vast negative influence on the environment. Nowadays green computing is hot topic especially when it comes to software development. Researchers mainly focus on efficient solutions to achieve energy efficient as a feature for computer hardware. Furthermore, energy efficient in software development life cycle can be of valuable importance in achieving green computing. Software can influence the environment indirectly by running in hardware that causes for example, carbon dioxide emissions.

As results, the researchers focus on the side of implementation of cloud components that affect energy consumption. Thus, components decide the type of cloud computing service models that are being used in the environment of cloud. Our efforts in this research project is on software implementation by elaborating in energy optimization at software or application level. Focusing on energy consumption of applications' profiling at runtime and narrow down hotspots iteratively; to achieve G-SDLC.

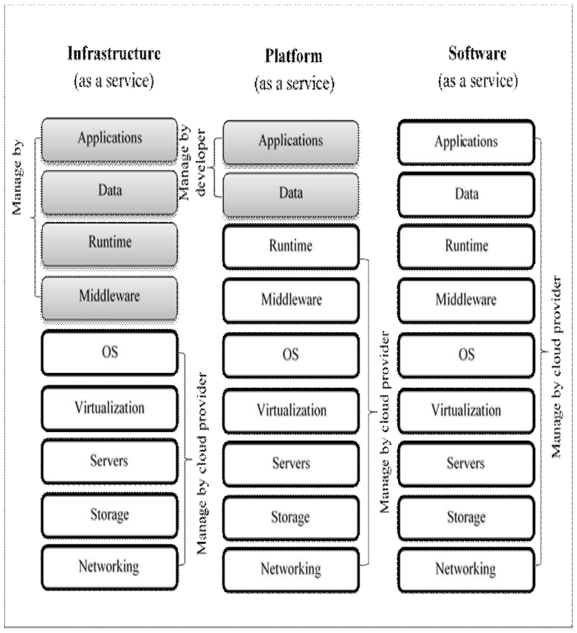


Figure 2: Services models of cloud [4]

Many efforts have been done to obtain green cloud framework covered virtualization mechanism and data center operations. Figure 2, shows the service models of cloud and the controller for each component [4]. Some efforts proposed green cloud framework for virtualization include virtual machine image management, Virtual machine (VM) scheduling and advanced data center design. Two areas in framework can show the improvements by originating efficient scheduling system for VM leads to minimize the operating costs of the cloud. Thus, an optimization of power would be achieved on server equipment or as reduction of data center temperature. This would reduce the energy required for cooling system as well and contribute in saving overall energy. Other general efforts, avoid such a waste of power, by automatic shutdown for idle software with saving its status and other shutdown for hardware no longer in use.

Developers need to understand the estimation of the program energy consumption to avoid the power waste in early stage of development. In the current research G-SDLC focuses on elucidate of green software development life cycle, especially in software implementation and parameterized methodologies to contribute in reducing energy consumption when it is running on cloud. The fact is that when the developers improve the source code with the help of parameterized phase; energy optimization of a software can be achieved by determining the part of code that consume high energy during the runtime. In alignment with regular five software development life cycle phases that covers requirement specification and analysis, design, implementation or coding, testing and maintenance. Our enhancement is adding parameterized development phase which focus on code generated in implementation phase. As stated, the source code plays a vital role in achieving energy efficient software. Therefore, all efforts focused on unnecessary use of controls, libraries and some unnecessary instructions by applying the parameters to reduce the energy consumption of software when running in cloud environment.

2. Literature Review

This section provides various solutions proposed by researchers to optimize energy conservation at hardware and software levels with different perspective. In reference [2] proposed typical SDLC with vital role to build efficient software in a systematic way by detecting energy-saving prospects in a typical SDLC to make software application eco-friendly in cloud. A framework of green cloud had been designed that consists of two side work in parallel to achieve green cloud. The first side is about the Service Level Agreement (SLA) in which the level of service the cloud customer expected to get from cloud providers. Both have to agree upon the required and high performance software by outline SLA terms relating to energy usage limits and responsibilities. The other side, concern about cloud service models and different approaches to optimize the energy of cloud components. In addition to some devices that required physical facilities. The paper illustrates how virtualization play significant role with scheduling schemes in cloud. These techniques have the opportunity to reduce energy cost and consumption through efficient resource use. Virtualization allows for resource utilization, live migration of VM, that all can minimize the cost of the energy [5] [6].

Reference [7] discusses the opportunities and challenges in energy saving for cloud data center. They illustrated that energy saving can be accomplished by turning servers not in use into lower energy mode, and by increasing the utilization of servers that already active. The approaches discussed include workload prediction; the concept of this approach demonstrated two mode of server that (turned on) based on time duration. The experiment showed that the server being in idle mode and turned on (while not needed) will consume much greater than amount of energy instead of just keeping it active. If the server in idle mode and not needed for a long period, then the power to be saved by way of turning it off can be higher than that to be consumed to turn the idle server back to be active. In addition, user will notice some undesired delay during this switching. This approach used to estimate cloud workloads in future and decide whether and when turning the server modes. The other approach is VM placement. Cloud centers consist of multiple clusters that are distributed in remotely geographic locations. Cloud providers receive VM request, then the cloud scheduler has to decide what a proper cluster should host and take care of the submitted request, with the cheapest power prices and highest confidence on green sources of energy in order maintain the efficient energy and reduce carbon emissions. These approaches manage entire cloud centers, which rely on virtualization capable to make cloud datacenters more energy efficient.

In reference [8] had built two levels of green software model and software parameters promoting green and environmentally manageable software. The first level covers the life cycle of a software development. A green software engineering process implemented with hybrid process taken the advantages of developing software from waterfall, agile and iterative development processes. Each stage evaluated with metrics to measure the greenness to produce a green stage. The second level, showed how the software with certain parameters make the software itself as tools used to assist in green computing in energy efficient manner. Such as use cloud computing with model of Infrastructure as a Service, to get the full control while developing the software and use of a reusable system and limit some features to achieve energy efficiency.

In reference [9] framework of energy efficient software development life cycle consisted of six phases similar to tradition SDLC phases: “Requirement phase”, “Design phase”, “Coding phase”, “Testing phase”, and “Maintenance phase” all linked in cyclic manner with centralized phase named “Green analysis phase”. This means that, when one phase is completed, then each phase connected to green analysis phase to check whether or not the particular phase follow the rules of green computing. The green analysis phase contain different concept of green computing such as green scheduler and compilers, grid computing, cloud computing, fault tolerance and other concept.

In reference [10] presented framework that provide scalable architecture of cloud computing to minimize the temperature within the data center. It covers two main parts virtualization and data center design. In virtualization, the target is to reduce data center temperature based on functioning of VM in cloud computing. This can be done by deriving efficient scheduling system for VMs. The scheduling process addresses the location of VMs within the cloud infrastructure while reducing operating costs of the loud itself. This is achieved by optimizing either server equipment power or the optimizing overall temperature within the data center. In data center design, the cooling system used is consisting of efficient Air Conditioners (AC) with more efficient power supplies to cool the server area which leads to optimize the power to operate the AC as well as the temperature within the data center.

The proposed work in [11], [12] illustrates different approaches for developing green software. Energy efficient SDLC for cloud applications is affected by both the concepts of cloud computing and green computing. The worldwide issue with any technology is energy conservation. Energy wastage can occur in both hardware and software levels. Different techniques are used to minimize energy usages as a green computing approach at cloud component and software engineering development.

3. Proposed System

The aim of this work to propose and investigate the framework of software development life cycle that introduces a new phase called parameterized development phase to enhance the energy efficiency of the software as shown in Figure 3, which helps the developers to optimize energy efficiency of software applications when it is being developed. Parameterized phase improves the software code by generating energy efficient code and to get efficient software which improves the source code for optimizing energy consumption and to determine the highest power consumption part in code. Therefore, the software application would be environment-friendly in cloud and same functionalities with other aspects of quality still maintained. Which is of our interest in achieving and optimizing energy efficiency by implementing G-SDLC in cloud computing environment.

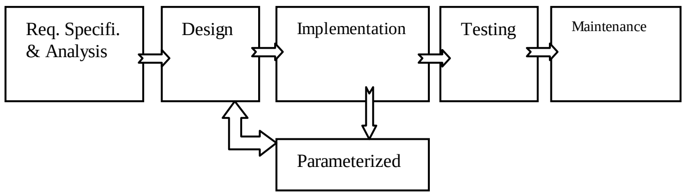


Figure 3: Green Software Development Life Cycle

Parameterized phase provides guidelines and appropriate methodologies for developers that promote software’s implementation phase. Set of parameters works as checklist that help in software code development. Such as data structure design, architectural design, abstract specification, and algorithm design. Applying the parameters should be based upon programming language used, application function and engineering hardware architecture. All these should be presented in earlier phase of software’s specification and requirement. Thus, developers can modify those parts consuming the energy instead of re-writing the code from scratch. Energy optimization comes with best practice and experience; developers will be able to identify the part of the code demand a particular pattern to be modified.

Application development depends on of the three model of cloud provided. The cloud provider, in case of software as a service (SaaS) is responsible for software development while it only provides a platform in case of platform as a service (PaaS) to allow developing an application. Whilst in an infrastructure as a service (IaaS) model, customers are able to take the all responsibilities of application development and maintenance in cloud as shown in Figure 2.The proposed research is designed as initial energy optimization model to be deployed on IaaS which allows full control for the user to manage the services remotely and other resource. IaaS allows the control over operating systems, storage, applications deployed and possibility of selecting network components [3]. This will help the developer later to manage the resources along with the optimized website/application. Thus, it will work in parallel with the optimized website/application to contribute in reducing the energy consumption expected from other perspective of cloud components.

Efficient code can be cost effective using different programming techniques that are used as parameters to adopt on source code. As result, developers can undertake code manually. Such as loop unrolling mechanism by executing the instructions within the loop in parallel. This will optimize compiler time as well as the energy required to compile the sequence instruction of loop. For example, if there is a program that deleting 100 items. Simply, For loop can accomplish this task by calling function, delete ( ) as shows in Table 1. As a result of this adaptation, the enhancement with “loop unrolling” only 20 iterations of the jumps and conditional branches need to be run, instead of 100. Developers are responsible to choose loop control variables and number of operations inside the unrolled loop structure. So, the output is same as the original code produced.

Table 1. Difference between normal loop and loop unrolling

Normal For-loop	**After applying loop unrolling**

int item;
for(item=0; item<100; item++)
{
delete(item);
}
	int item;
	for(item=0; item <100; item+=5)
	{
	delete(item);
	delete(item + 1);
	delete(item + 2);
	delete(item + 3);
	delete(item + 4);}

The scope and declaration of variables, variable types, nested loops, switch statements and are other programming techniques can influence processing as well as energy consumption. For example, int variable can be stored in form of double, whereas in this case it will use 8 bytes of memory instead of 4 bytes as if stored as int. All these techniques which apply to cloud applications targeting energy saving. In the cloud environment, applications may be needed to share parameters and communicate across the networks. In order to optimize energy consumption, number of parameters must be controlled during information transmission. The programming language can affect energy consumption of application. As few programming languages are effective in use of memory and CPU by applying techniques of garbage collection and multithreading. For example, ASP.NET garbage collector provides good use of memory with high speed allocation services. It preforms minimization that will give developer much less than optimal performance and keep the memory usage effective [13].

4. ENERGY OPTIMIZATION IN CLOUD COMPUTING

Today, energy efficiency is the most vital constraints for cloud computing. It decides the operational costs and the benefits of capital investment. Since CC depend on data center industry deployed worldwide. Cloud applications optimization will not only reduces the data centers energy consumption, rather also will contribute in optimize overall cloud computing systems globally. In [14], issues are addressed in regards to reducing energy consumption of data centers, as well as in cloud computing. They include but not limited to virtual machines placement workload scheduling or load balancing as illustrated below:

Virtual Machines placement: Virtualization is an efficient operation in cloud data centers. The physical equipment that is energy consumption can be replaced virtually in data center. VMs consolidation strategies try to accommodate lowest possible number of physical machines to host a certain number of VMs. In virtualized environment, scheduling and load balancing are some techniques used effectively to manage operation in data centers. VM is ensuring efficient utilization of storage to serve hosted application workloads and then lead to minimize energy consumption in data center.

Workload scheduling: It contributes in improving the efficiency of data center in a way that specifies the resources that would be used to complete the work. The work could be data flow, processes or threads that are in turn waiting to being processed. The focus is to achieve efficient utilization of resources selected from the server with cost and energy effective. Additionally, energy efficient workload scheduling tends to ponder all active workloads on a minimum set of servers with least possible communication resources.

Load balancing: Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload for any particular resource. Load balancing can efficiently distribute the incoming traffic among available servers in fast and reliable manner. It ensures that no one server has overloaded more than other, that may cause it to be down. As the inactive server (in case of shutdown) require energy to turn it on, and then heat is generated more and cooling system will consume electrical energy to keep good temperature. Load balancing is the important enabler for saving energy by avoiding unnecessary power consumption. However, recoveries after the server turned ON will causes some delays.

In regards to energy efficiency of SDLC in cloud computing focused on optimizing the code that cause's energy consumption which can be achieved by applying a list of parameter proposed to the developer and help in writing efficient code. In this section first the code generation stages are discussed, and various effective parameters (techniques) to optimize the generated code are illustrated.

A. Code Generation

Programmers at implementation phase are recommended to optimize power as much as possible in all components of software. Subsequently, energy efficient software can be delivered. Code generation can be defined as the process of generating target instructions to implement the source code [15]. It is the translation of UML diagrams into executable code. In code generation, the compiler actually converts intermediate representation that is data structure or internal code used in compiler, into executable source code. Then, the compiler maps the source code with registers (storage allocation) or memory locations for each targeted variables/symbols [16]. This will produce a set of instructions that perform the function of developed software that are spaced and time efficient.

Lexical analysis is the first phase of compiler. It scans the characters to make up the stream of code and produce lexemes that are meaningful sequence (token stream). The second phase is syntax analyzer (parsing), it use the components produced by lexical analyzer (token stream) to create intermediate representation in form of tree-like. This representation represents the operation as node and the leaves as arguments of the operation. In semantic analyzer phase, the meaning of structures of the syntax-tree formed in syntax analyzer phase is analyzed and verified. It checks the authority of semantic. Therefore, each operator has matching operands. At code generation phase, the targeted machine code is generated. It may include optimization of code, operations and structure.

1) Major Tasks in Code Generation

There are some major tasks that are typically part of code generation phases. These tasks should be taken in account at earlier stage to achieve energy efficient code. Register is the preferred location for operands. It makes the instructions execution faster since the fetched operands reside on the specified registers. Despite the fact, there is a need in some circumstances to assign memory location due to registers scarcity. There are major tasks illustrated in [17] that related to registers in code generation:

a) *Register allocation:* It is a phase of code generation, responsible for deciding which value can most profitably be held in registers and at what points in program. Operands accessed from registers require less energy than those that accessed from memory. In some cases instructions cannot access memory directly, so the operands explicitly loaded to register for use.

b) Instruction scheduling: It determines the order of instruction to be performed. It has critical effect on speeding the optimization on pipelined machines. It is important to ensure that the instruction scheduler handle the delay-load of instructions that completed with several cycles. So, only load the instructions that facing the pipeline.

c) Instruction selection: It is the phase where the instruction has been selected to perform the required computation. Instruction selection is the process of selecting the best instruction to perform computation. In other words, what instruction to use. The selected instruction should be able to perform the computation as fast as possible, combined well with the surrounded instruction of computation. The selector must choose the best instruction that can produce the best code with minimum size and execution time. A good code generator is easier to implement, test and maintain. Thus, careful assignment of variables and expressions to registers lead to increase the efficiency of compiler.

B. Code Optimization

The term optimization means that the compiler produces code which is more efficient than the obvious code. Since there is no way that guaranteed that the code produced by a compiler is faster than any other code that performs the same task. Therefore, developer intervention is required to enhance the source code. Code optimization is one of the main goals to achieve energy efficient software. It is the heart of the proposed parameterized development phase. It attempts to improve the code, so that the code occupies less space, executes fast and consumes less power. Optimization of code is done at compilation stage by applying set of the proposed parameters without influencing existing coding.

To get better code at code generation stage; developer recommended to reuse already computed values (avoiding extra computation). Since the reuse is less expensive than re-computing. More efficiency can be achieved by avoiding unnecessary stores of variables, extra loads of memory and register-to-register moves. Minimizing the movement of data can save the energy by writing software with appropriate data structures. Therefore, all efforts will work on unnecessary use of controls, libraries and some unnecessary instructions; by applying the parameters to reduce the energy consumption of software when running in cloud environment. There are plenty of approaches to observe the code parts that influence more in the total energy consumption (i.e. energy hotspots). Different programming techniques are established for code level optimizations and used as parameters that developers undertaken them manually.

1) Code Optimization Techniques

Code optimization techniques target to reduce execution time and reduce space and consume less energy. There are several techniques which can help a software run faster. These techniques aim to improve the code in a way, so that CPU time and memory are optimized. The goals of code optimization are to reduce the execution time, memory consumption and remove unnecessary/unreachable or redundant code without affecting the output of code. Below are various methods of manual optimization that often yield to improve the source code. These techniques include but not limited to constant propagation, constant folding, strength reduction, common sub expression elimination, unreachable code elimination/dead code elimination and loop optimization.

a) Constant propagation: Developer recommended replacing the variable by its constant value, without creating declaration/definition statement for variable. Directly propagate the constant value at use of the variable in code.

Original code	Transformed code

int N = 10; int C = 2;
for (int i =0; i<N; i++)
{
*s = s + i*C;*
}
	for (int i =0; i<10; i++)
	{
	*s = s + i*2;*
	}

b) Strength reduction operation: Replacing an expensive operation with a cheaper operation that can produce same results. It replaces slow mathematical operations with faster one.

Original code	Transformed code
int y = 2 ; *int x = y^3;*	*int y = 2 ;* *int x = y*y*y;*

c) *Constant folding:* It is similar to constant propagation. However, in constant folding, constant expressions evaluated and recognized at compile time better than that of computing them at run time. So, constant folding delivers the constant of an expression. It considered as algebraic simplification method.

Original code	Transformed code

int a = 30;
int b = 9 - (a / 5);
int c;
*c = b * 4;*
if (c > 10) {
c = c - 10;
*} return c * (60 / a);*
	int a = 30;
	int b = 3;
	int c;
	*c = b * 4;*
	if (c > 10) {
	c = c - 10;
	*} return c * 2;*

d) Common sub expression elimination: In the body of code i an expression repeated more than once, it is better to have common sub expression. Common sub expression can be local within single basic block or global on entire procedure. (Constraint: the value of the expression and its operands should not change across various occurrences).

Original code	Transformed code

*a = (b + c)*m;*
x = b + c;
*y = (b + c) * z;*
	*Tmp = b + c; a = Tmp *m;*
	x = Tmp;
	*y = Tmp * z;*

e) *Dead code elimination/Unreachable code elimination:* Unreachable code is a fragment of the source code that can never be executed and also known as dead code as there is no control flow path exist to make it executed. The optimization can be in this case by removing the portion that does not affect the function results. The definition int Z = X * Y; is certainly not reached and executed as the function returns before the definition is get hold of. Thus, Z definition can be castoff.

Original code	Transformed code

int addition (int X, int Y)
{
return X + Y;
*int Z = X * Y;*
}
	int addition (int X, int Y)
	{
	return X + Y;
	}

f) Loop Optimization: Loop optimization is the process of enhancing the execution speed of the loop and reducing the number of iteration. Loop optimization can improve the performance of the cache as most of the spaces and time spent on loop. Many techniques are established to consume less energy and make the execution of the loop faster. These techniques include loop invariant detection and code motion, loop unrolling and loop fusion.

g) Loop invariant detection and code motion: If the body of the loop consists of expressions or statements, it is better to move them outside loop's body. This would improve efficiency by computing specific value once before the loop starts. The calculation x = y + z and x * x are moved outside the loop.

Original code	Transformed code

for (int i = 0; i < n; i++) {
x = y + z;
*a[i] = 6 * i + x * x;*
}
	x = y + z;
	*t1 = x * x;*
	for (int i = 0; i < n; i++) {
	*a[i] = 6 * i + t1;*
	}

h) Loop unrolling: It is a technique of optimization in which increasing the speed of program by eliminating or reducing instructions that control the loop. Developer suggested injecting body of the loop with some expression, in direction to make shortcut of the condition execution.

Original code	Transformed code

int item;
for(item=0;item<100;
item++)
{
delete(item);
}
	int item;
	for(item=0;item<100;item+=5
	)
	{ delete(item);
	delete(item + 1);
	delete(item + 2);
	delete(item + 3);
	delete(item + 4);
	}

i) Loop fusion: It is the process of replacing multiple loops that have same iteration over same identified range with single loop.

Original code	Transformed code

int i, a[100], b[100];
for (i = 0; i < 100; i++)
a[i] = 1;
for (i = 0; i < 100; i++)
b[i] = 2;
	int i, a[100], b[100];
	for (i = 0; i < 100; i++)
	{ a[i] = 1;
	b[i] = 2;
	}

5. RESULTS

In this section, the factors affecting the energy consumption of code are illustrated. CPU utilization and memory allocation are the factors that impacted the way of implementing and designing the software application. Performance profiler tool embedded in Microsoft Visual Studio 2017 is used to measure the performance of code in local host. Two experiments are proposed in their original status and after applying the suitable parameter. Further, the performance of code with respect to CPU utilization and memory allocation is measured.

A. Code Performance Measurement

The energy consumption depends on the application itself. If the application is long running with high CPU and memory requirements then its execution will result in high energy consumption. Therefore, energy consumption will be directly proportional to the application's profile. Developer must put more efforts in regards to optimize the performance of code. Performance optimization is endless process. There is always an opportunity for the developer to improve and make the code run fast in efficient manner. Hence, software should be developed to be energy efficient either at earlier stage or during runtime.

The measurement of these parameters is the roadmap to know whether the software has good or low performance in terms of CPU utilization and memory it consumes. In other words, efficient application has good performance when its execution takes less time. However, inefficient application has low performance that leads to higher CPU and memory utilization, then higher energy consumption more time to execute. Object-Oriented Programming Language most probably provides integrated garbage collection that helps in free the developer from dealing with memory de-allocation. However, garbage collection may consume resources in computation to take decision of which memory to free. Therefore, this could lead to decreased performance of the code [18]. Manual optimization remains mostly important at the source code level. Developer can use his/ her experience and knowledge to perform optimization in creatively manner and obtain good performance [19].

Optimizing the code will indicate good performance in regards to CPU and memory utilization. This can be measured by using performance profiler to identify code bottleneck. The code bottleneck can be found, for example, in loop that is executed hundreds/ thousands of times. Developer can redesign the body of loop in order to form in way that does not need to be executed hundreds/thousands of times [20].

1) Measure the Performance Using Performance Profiler Tool in Microsoft Visual Studio 2017

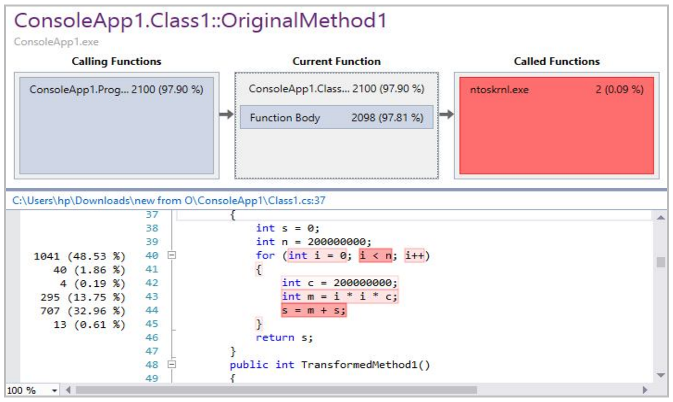
Microsoft Visual Studio 2017 profiling tool, allows developer to analyze performance issues in their application code that is written with any programming languages of ASP.NET. It reports the performance from different perspective, including CPU usage, memory usage, performance wizard and others. It reports the performance of code by collecting information about the functions that performing work. Furthermore, provides timeline graph the developer can use to focus on specific segments of code [21]. Measuring the performance of code can be done using performance profiler in two steps: collect CPU/memory usage data and analyze CPU/memory usage data. To measure the time CPU is spending the CPU usage tool are used so that the developer can modify the code accordingly. Similarly, memory allocation shows the total memory allocated to execute the subject code [22] [23].

2) Performance Optimization Experiments Using Proposed Parameters

One cannot really know about performance of code without measuring it. However, the function code of an application can be measured to see how long it takes and memory occupies. The good performance is defined by the function that taking less time to be executed with less memory utilization. On other hand, the low performance is defined by function that taking more time to be executed and consuming the system resources. The various Code Optimization techniques/parameters introduced in the earlier section are performed on two experiments discussed below by applying these parameters to observe how memory and CPU usage is optimized to enhance the source code and makes it performs efficiently. These experiments illustrate the performance of the original piece of code compared to the performance of the transformed code. For ease, different names given for two fragments of code "Original Method1 and Original Method2" that are the codes need to be improved, and the other fragments named by "TransformedMethod1 and TransformedMethod2" which are manually enhanced code.

Experiment 1: Constant Propagation

The OriginalMethod1 () is piece of code written to perform some calculation within the body of for loop. The calculation operated using values assigned to variables which required accessing the registers to read their values.


ConsoleApp1.Class1::OriginalMethod1
ConsoleApp1.exe
Calling Functions
Current Function
Called Functions
ConsoleApp1.Prog... 2100 (97.90 %)
ConsoleApp1.Class... 2100 (97.90 %)
Function Body 2098 (97.81 %)
ntoskrnl.exe 2 (0.09 %)
100 %

C:\Users\hp\Downloads\new from O\ConsoleApp1\Class1.cs:37

```
                        {
                            int s = 0;
                            int n = 200000000;
1041 (48.53 %)              for (int i = 0; i < n; i++)
  40 (1.86 %)               {
   4 (0.19 %)                   int c = 200000000;
 295 (13.75 %)                  int m = i * i * c;
 707 (32.96 %)                  s = m + s;
  13 (0.61 %)               }
                            return s;
                        }
                        public int TransformedMethod1()
                        {
```

Figure 4: Hot Spot of OriginalMethod1

Now let’s evaluate the OriginalMethod1 () performance side. It is recommend to start analyzing the code by evaluating the CPU usage and identifying the peak of the CPU usage. This means that the peak point is the part of code that you need to work on it, in order to improve it. Using the performance profiler tool in Microsoft Visual Studio, Figure 4, highlights parts of code that have a chance to be improved by developer. These highlights can be accessed by investigating the peak point reported by CPU usage or running time addressed in Figure 4. Moreover, Figure 5 illustrates the overall percentage of CPU usage or running time of code execution with total of 2,145 samples. The sampling is recommended method for starting the investigation of code and is useful for finding issue of processor utilization [24].

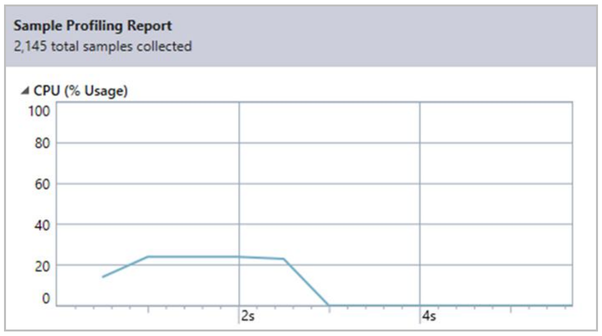


Figure 5: CPU usage of OriginalMethod1

Figure 6, is snapshot of the total memory allocated for OriginalMethod1 (). It is executed with total of 7,949 bytes of memory allocation as per the memory profiling report.

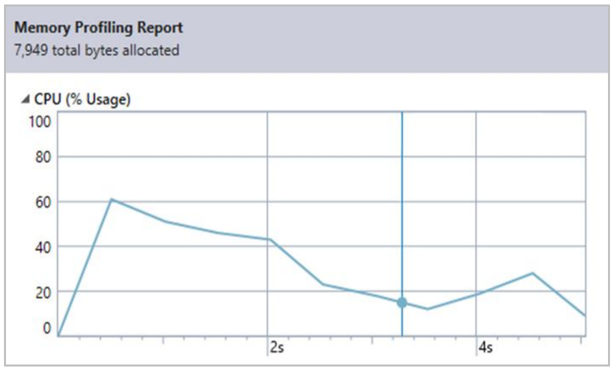


Figure 6: Total memory allocated for OriginalMethod1 ()

The OriginalMethod1() is completely run in 2,518 seconds as illustrated in Figure 7, as per instrumentation profiling report.

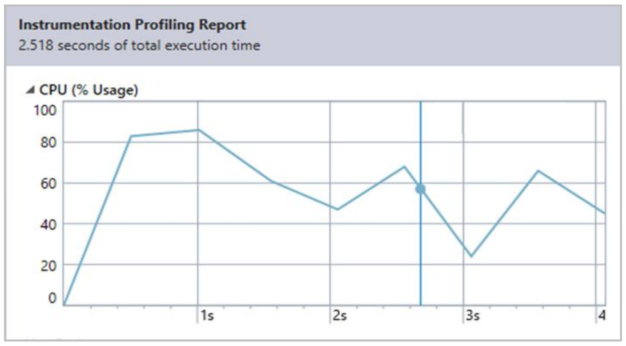


Figure 7: Total execution time of Original Method1

After getting the above performance data, developer manually applied the suggested parameter which transforms the code in a way that the performance optimized. In TransformedMethod1 (), the value of the constant propagated directly in expression as shown below:

```
public int TransformedMethod1()
 {    int s = 0;
   for (int i = 0; i < 200000000; i++)
   {     s += ((i * i) * 200000000);
    }           return s;
```

Performance evaluation procedures applied to TransformedMethod1 (). The results showed an improvement of CPU usage which reduces the total number of the collected samples to 1,690 as illustrated in Figure 8. This is an improvement of 21% compared to the OriginalMethod1 ().

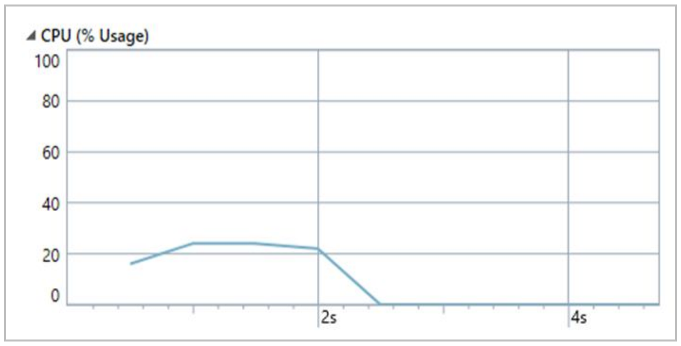


Figure 8: Total CPU usage of TransformedlMethod1

Furthermore, in Figure 9, the total memory allocated for the TransformedMethod1() was 7,937 bytes which shows the decrease of 0.15 % compared to the OriginalMethod1 (). It is very slightly decrease but means a lot for the developer. In Figure 10, the total execution time reported by instrumentation profiling is 2,044 seconds. This means that TransformedMethod1() takes less time than the OriginalMethod1 () to be completed. Thus, it becomes light code.

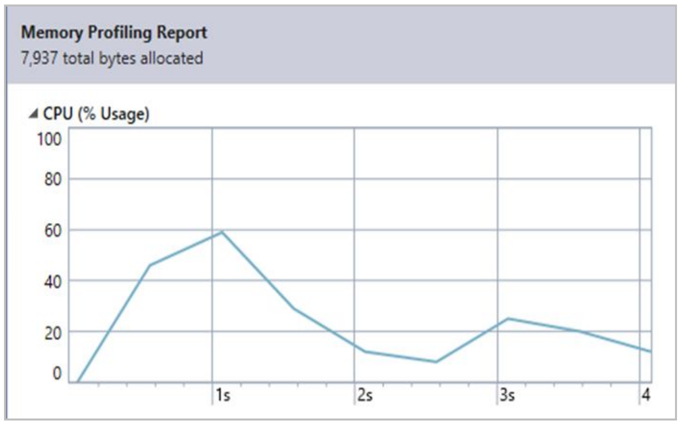

Memory Profiling Report
7,937 total bytes allocated

Figure 9: Total memory allocation of TransformedlMethod1

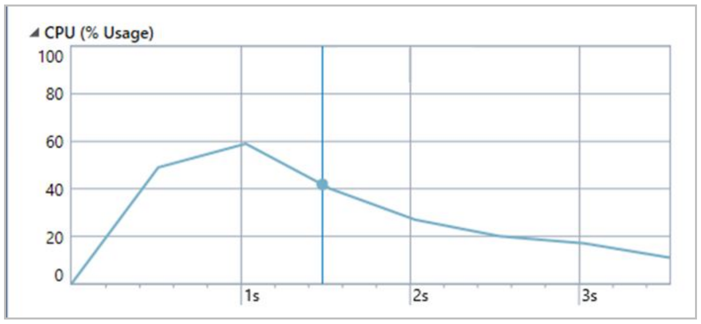


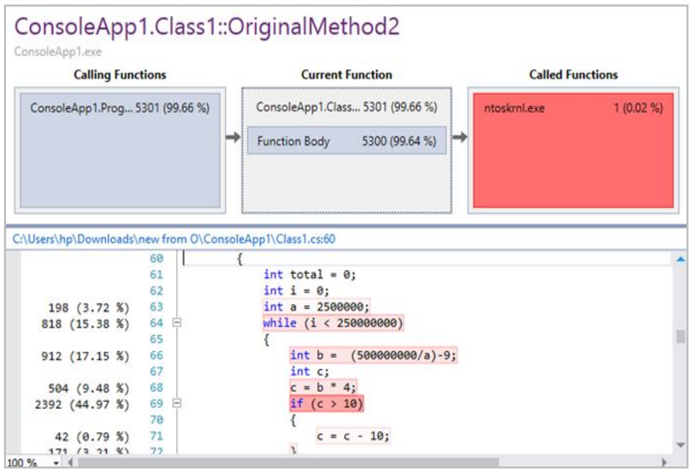
Figure 10: Total execution time of TransformedlMethod1

Experiment 2: Constant Folding

In this experiment, the piece of code is evaluated in which the body of while loop handled some calculation as shown below:

```
public int OriginalMethod2()
  {  int total = 0;
   int i = 0;
   int a = 2500000;
   while (i < 250000000)
    {
     int b =  (500000000/a)-9;
     int c;
     c = b * 4;
     if (c > 10)
     {  c = c - 10;
      }
     i++;
    total = c;
return total ;
}
```

Same evaluation procedure proposed on experiment 1 are followed in this experiment. Figure 11 shows the part of code that reported by CPU usage as peak point of OriginalMethod2 () execution. This means that the developer should modify these highlighted parts.


ConsoleApp1.Class1::OriginalMethod2
ConsoleApp1.exe

Calling Functions	Current Function	Called Functions
ConsoleApp1.Prog... 5301 (99.66 %)	ConsoleApp1.Class... 5301 (99.66 %)	ntoskrnl.exe 1 (0.02 %)
	Function Body 5300 (99.64 %)	

C:\Users\hp\Downloads\new from O\ConsoleApp1\Class1.cs:60

```
                     {
                         int total = 0;
                         int i = 0;
  198 (3.72 %)           int a = 2500000;
  818 (15.38 %)          while (i < 250000000)
                         {
  912 (17.15 %)              int b =  (500000000/a)-9;
                             int c;
  504 (9.48 %)               c = b * 4;
 2392 (44.97 %)              if (c > 10)
                             {
   42 (0.79 %)                   c = c - 10;
  171 (3.21 %)               }
```

Figure 11: Hot spot of OriginalMethod2

Figure 12, illustrates the overall percentage of CPU usage or running time of code execution with total of 5,435 samples were collected as per the sample profiling report.

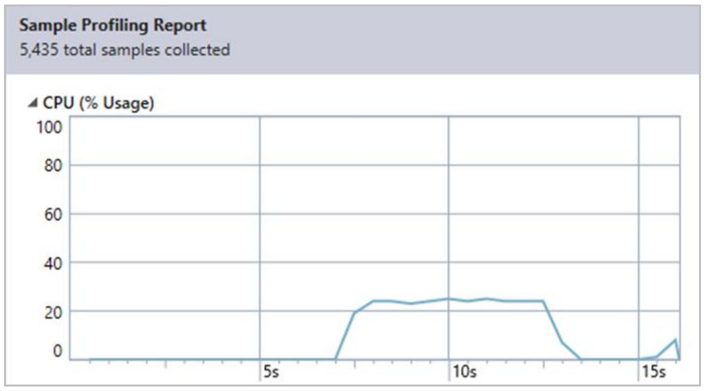

Sample Profiling Report
5,435 total samples collected

Figure 12: CPU usage of OriginalMethod2

Figure 13, illustrates that OriginalMethod2 () executed with total memory allocation of 7,949 bytes as per the memory profiling report.

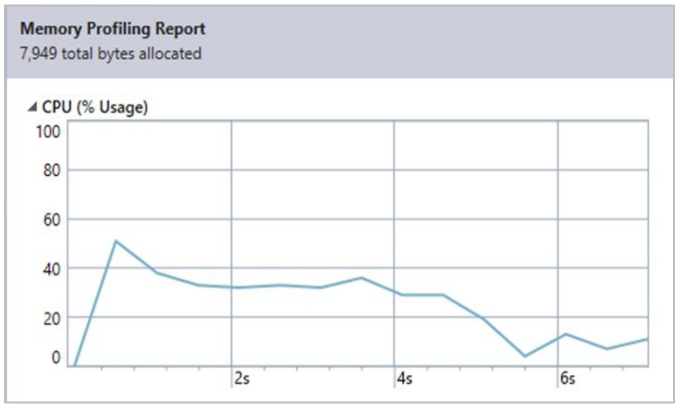

Memory Profiling Report
7,949 total bytes allocated

Figure 13: Memory allocation of OriginalMethod2

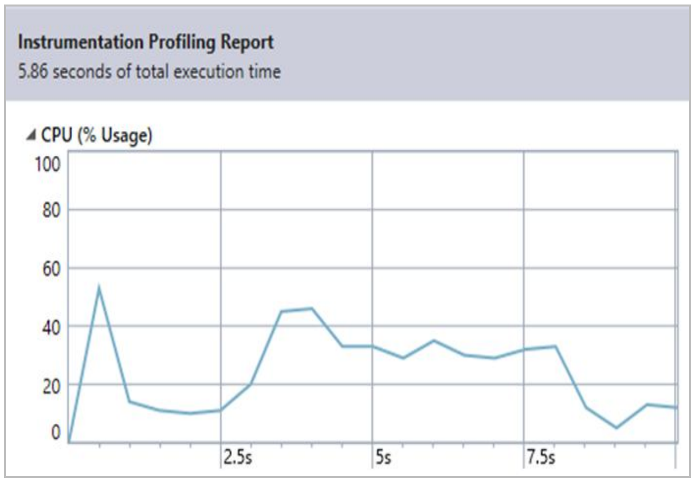


Figure 14: Total execution time of OriginalMethod2

The OriginalMethod2() is completely run in 5.86 seconds as illustrated in Figure 14, based on instrumentation profiling report. Instead of computing the values at run time, the developer manually calculated the expression and then constant folding can be delivered directly as follow:

```
public int TransformedMethod2()
{
    int i = 0;
    int c = 0;
    while (i < 250000000)
    {
        c = 764 - 10;
        i++;
    }
    return c;
}
```

A significant improvement is observed on Figure 15, as the line of code in the TransformedMethod2() becomes less. This yield that 1,782 total samples were collected which is less than OriginalMethod2() by 67%.

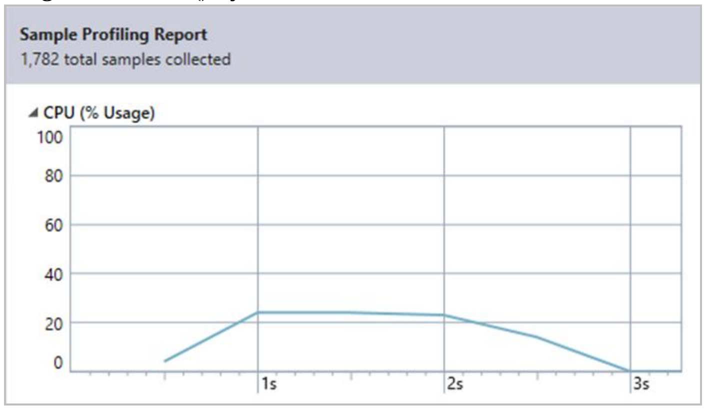


Figure 15: Total CPU usage of TransformedlMethod2

Moreover, 7,937 bytes are the total memory allocated after the modification which is slightly less than the OriginalMethod2() by 0.15%. Figure 16, illustrates the memory profiling report for TransformedMethod2().

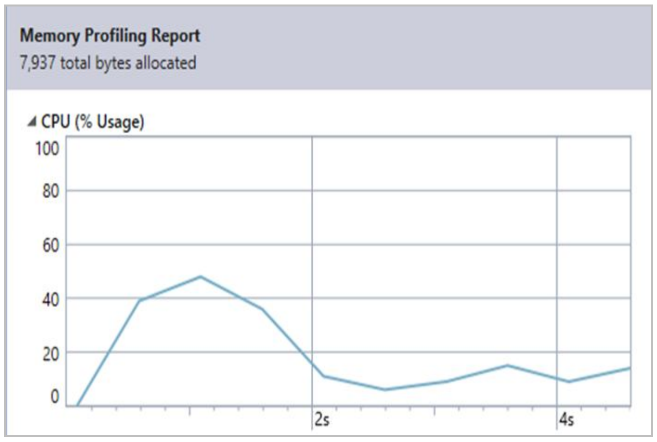


Figure 16: Total memory allocation of TransformedlMethod2

Overall, the improvement can be observed in execution time. TransformedMethod2() takes 1.507 seconds an improvement of 74% compared to OriginalMethod2() as showed in Figure 17.

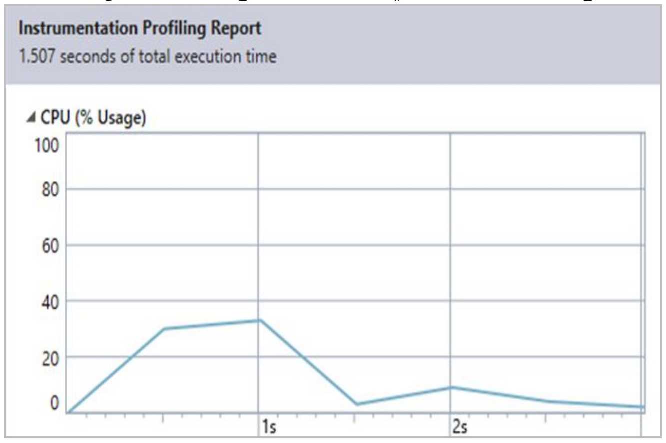


Figure 17: Total execution time of TransformedlMethod2

Note: CPU (Usage %) shows in memory allocation is different from the one showed in CPU usage analyzer. The reason is that the CPU usage analyzer shows the percentage of CPU time with samples collected and spent in calling and the called functions [24]. Since the proposed experiments are called in console class.

```
class Program
    {
        static void Main(string[] args)
        {
            Class1 c1 = new Class1();
            c1.OriginalMethod1();
            Console.WriteLine( "Press ‘Enter’ to start
analysis..");
             Console.ReadKey();
        }
    }
```

However, the CPU (Usage %) shows in memory allocation is referred to the percentage of CPU time spent in the function body only [22]. In addition, the actual reported values will be different depending on the machine environment.

3) *Summary of Experiments Analysis*

The summary and the result obtained after implementing proposed parameters/optimization techniques. Table 2, shows comparison of total execution time of two experiments with improvement of 18% and 74%, respectively which indicates that the code becomes fast and takes less time in execution.

Table 2: CPU execution time improvement

	Original code CPU execution time (Seconds)	Trans. code execution time (Seconds)	Performance improvement (%)
Experiment 1	2.518	2.044	18
Experiment 2	5.86	1.507	74

Table 3, demonstrates the improvement of total memory allocation. It slightly decrease because of the small fragment of code. Hence, energy can be saved by reducing data movement. This can be achieved by using appropriate methodology of data structures, understanding and exploiting the underlying memory hierarchy. Moreover, it can be achieved by designing multi-threaded code that reduces data movement then the memory allocation will be reduced [25].

Table 3: Memory performance improvement

	Original code memory allocation (Bytes)	Trans. code memory allocation (Bytes)	Performance improvement (%)
Experiment 1	7,949	7,937	0.15
Experiment 2	7,949	7,937	0.15

An improvement in CPU execution time and memory usage is evidenced in the performance of codes using the proposed parameters in SDLC. This can be observed clearly in Figure 18 and 19, respectively. Figure 18, shows the improvement on CPU utilization. Despite the fact that, the differences are slight but have impact in code execution. Memory allocation decreases in both experiments as reflection of performance optimization. Thus, the performance optimization will be more evident across entire website/application developed with G-SDLC.

6. Conclusion

Software has effect on cloud environment same as hardware despite that it is indirect. Cloud architecture and its component are not the only factors to be managed to achieve energy efficiency environment; software running on cloud with taken into account the software development life cycle will play significant role to achieve this target. The developer with advanced programming skills is in the first line of the process of getting energy efficient software. Performance profiler tools helps one to identify the hot spot/part of code that consumes the resources and time of execution. The two experiments proposed are clearly presenting the improvement in performance optimization on local machine.

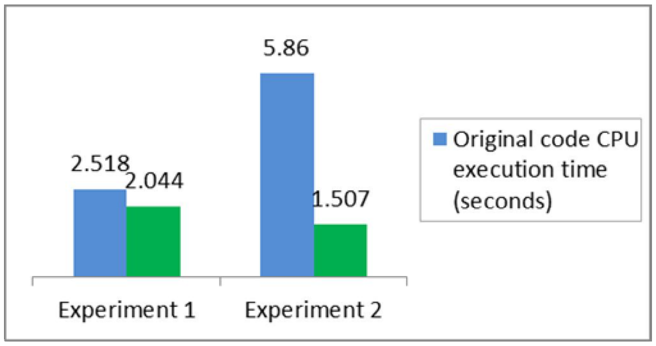


Figure 19: CPU execution time improvement

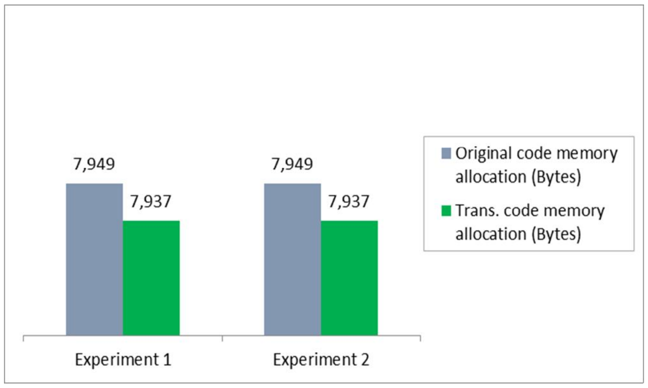


Figure 20: Memory performance improvement

Software development improvement leads to reduce power costs. Therefore, getting green cloud and saving money. The latest processors have improved energy efficiency in hardware side. In software side, developer needs to contribute and follow appropriate parameters in implementation phase, as small change to software engineering process can make a large difference in regards to energy.

REFERENCES

1. A AlNuaim and S Ahmed, Fog computing: **A novel approach to provide security in cloud computing**, *Indian Journal of Science and Technology*, Vol. 11, no. 15, April 2018.
2. Saurabh Kumar Garg and Rajkumar Buyya. 2012. **Green cloud computing and environmental sustainability**. *In Harnessing Green IT: Principles and Practices, San Murugesan and G. R. Gangadharan*. Wiley, UK, 315-340.
3. Nitin Singh Chauhan and Ashutosh Saxena, **A Green Software Development Life Cycle for Cloud Computing**, *IT Professional*. Vol.15, pp. 28-34, 2013.

4. Laghari, Asif Ali, Hui He, Imtiaz A. Halepoto, M. Sulleman Memon, and Sajida Parveen. **Analysis of Quality of Experience Frameworks for Cloud Computing**. *IJCSNS* 17, no. 12 (2017): 228
5. M. Nageswara Prasadhu, Dr.M.Mehfooza, **An Efficient Hybrid Load Balancing Algorithm for Heterogeneous Data Centers in Cloud Computing**, *International Journal of Advanced Trends in Computer Science and Engineering*, 9(3), May – June 2020, 3078 – 3085.
6. Syed.Karimunnisa, Dr.Vijaya Sri Kompalli, **Cloud Computing: Review on Recent Research Progress and Issues**, *International Journal of Advanced Trends in Computer Science and Engineering*, 8(2), March - April 2019, 216 – 223.
7. Mehiar Dabbagh, B. Hamdaoui, M. Guizani and Ammar Raye, **Toward Energy-Efficient Cloud Computing: Prediction, Consolidation and Over commitment**, *IEEE communications society*. Vol.29, pp. 56-61, 2015.
8. Sara S. Mahmoud, Imtiaz Ahmad, **A Green Model for Sustainable Software Engineering**, *International Journal of Software Engineering and Its Applications* Vol.7, pp. 55-74, 2013.
9. S. K. Sharma, P. K. Gupta, R. Malekian, **Energy efficient software development life cycle - An approach towards smart computing,** *2015 IEEE International Conference on Computer Graphics, Vision and Information Security* (CGVIS), Bhubaneswar, 2015, pp. 1-5
10. Andrew J. Younge et al., **Efficient Resource Management for Cloud Computing Environments**, *International Green Computing Conference*, Chicago, USA, 2010
11. H. Huang, **A Sustainable Systems Development Lifecycle**, *Pacific Asia Conference Information Systems* (PACIS 08), Stockton, USA, pp.1-12, 2008.
12. Bob Steigerwald and Abhishek Agrawal, **Developing Green Software** [Online]. Available: http://software.intel.com/en-us/articles/developing-green-software.
13. Rico Mariani (2003, April 30). **Garbage Collector Basics and Performance Hints** [Online]. Available: https://msdn.microsoft.com/en-us/library/ms973837.aspx.
14. Dzmitry Kliazovich et al., **Energy Consumption Optimization in Cloud Data Centers, in Cloud Services, Networking and Management,** 1st ed, New Jersey, 2000, pp 10-27.
15. Sathishkumar Udayanarayanan and Chaitali Chakrabarti, **Energy Efficient Code Generation for DSP56000 family**, *International Symposium Low Power Electronics and Design*, Rapallo, Italy, pp. 247-249, 2000.
16. Alfred V. Aho et al., **The Structure of a Compiler, in Compilers Principles, Techniques and Tools** 2nd ed. Boston, USA, pp.5-6, 2007.
17. Todd Alan Proebsting, **Code Generation Techniques**, Ph.D. dissertation, Dept. Computer Science, University of Wisconsin, Madison, USA, 1992.
18. Matthew Hertz and Emery D. Berger, **Quantifying the Performance of Garbage Collection vs. Explicit Memory Management,** San Diego, California, USA,Rep. 1-59593-031-0/05/0010, October 2005
19. Rudrik Upadhyay et al. (2017, September 7). **A Practical Approach to Optimize Code Implementation** [Online]. https://www.einfochips.com/wp-content/uploads/resources/a-practical-approach-to-optimize-code-implementation
20. Michael E. Lee, **Optimization of Computer Programs in C**, Ontek Corporation, Laguna Hills, USA, 1997.
21. Mike Jones et al. (2017, February 27). **Profile application performance in Visual Studio** [Online]. https://docs.microsoft.com/en-us/visualstudio/profiling/beginners-guide-to-performance-profiling?view=vs-2017.
22. Theano Petersen et al. (2018, April 11). **Analyze CPU usage** [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/profiling/cpu-usage?view=vs-2017.
23. Genevieve Warren et al. (2018, February 1). **Analyze memory usage** [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/profiling/analyze-memory-usage?view=vs-2017.
24. Mark McGee et al. (2017, February 27). **Beginners guide to CPU sampling** [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/profiling/beginners-guide-to-cpu-sampling?view=vs-2017.
25. K. Eder and J. P. Gallagher, **Energy-Aware Software Engineering, in ICT-Energy Concepts for Energy Efficiency and Sustainability**, G. Fagas, L. Gammaitoni, J. P. Gallagher, and D. J. Paul, Eds. Rijeka, Croatia: IntechOpen, 2017.