applied sciences

MDPI

Article

A Scalable Parallel Architecture Based on Many-Core Processors for Generating HTTP Traffic

Xinheng Wang *, Chuan Xu, Wenqiang Jin, Jiajie Wang, Qianyun Wang and Guofeng Zhao

Future Network Research Center, Chongqing University of Posts and Telecommunications, Chongqing 400065, China; xuchuan@cqupt.edu.cn (C.X.); jinwenqiang6668@gmail.com (W.J.); wangjiajie92@outlook.com (J.W.); wangqy25@outlook.com (Q.W.); zhaogf@cqupt.edu.cn (G.Z.)

* Correspondence: xhwangcqupt@gmail.com; Tel.: +86-133-4024-4910

Academic Editor: Lorenzo J. Tardón

Received: 8 December 2016; Accepted: 2 February 2017; Published: 8 February 2017

Abstract: The past years have witnessed the significant development of the Internet. Numerous emerging network architectures and protocols have triggered the demand for traffic generators which stand in stark contrast to previous schemes. Namely, fixed test content is inefficient in the presence of such a dynamic and realistic demand. Moreover, the requirement of high-performance has raised the stakes on developing a new concurrent system. In this paper, we present a hierarchical parallel design for a Web traffic generator on a TILERAGX36 processor, called TGMP. We discuss the challenges in developing its hierarchical architectural design, and elaborate on its implementation details. Specifically, in order to generate a realistic network workload over a long and large time scale, we propose a user-control scheme based on cubic spline interpolation. To better improve the scalability of the system and satisfy the required flow rate, we adopt techniques, including optimization of parameters under the Linux kernel, event-driven concurrency, and parallel architectures of a TILERAGX36 processor. The experimental results demonstrate that TGMP is able to create real traffic and simulate 50,000 users accessing the Web server simultaneously.

Keywords: scalability; high performance; TGMP; TILERAGX36

1. Introduction

A traffic generator is the major tool to provide network background traffic to evaluate and verify the performance, protocol, and security of the experimental network. With the continuous in-depth application of the emerging network architecture, such as SDN (Software Defined Network), and CCN (Content-Centric Networking), there has been an urgent and ever-increasing demand for a traffic generator with high-performance, flexibility, low cost and reality. Unfortunately, such a tool is not easily available, and is usually of much smaller scale than needed. Recent research suggests that the Web still occupies 15%∼18% of the traffic [1]. Therefore, generating high-performance and a representative Web traffic generator is crucial for evaluating the emerging network architecture under different network conditions.

At present, some commonly used tools have been made for a Web traffic generator, of which the two most typical types are the special equipment based on hardware and the software tool running on general hardware. The former is generally designed for a particular test scenario which lacks flexibility and extensibility, such as Spirent Avalanche [2], and Ixia IxLoad [3]. Spirent Avalanche is an application test which creates real-world dynamic user behavior with advanced browser-to-application-interaction capabilities. However, it is hard to cope with highly changeable scenarios. Owing to its flexibility and the simplicity of implementation, the latter provides low-cost and quick deployment. Some other well-known tools are Webstone [4], ProWGen [5] and GlobeTraff [6]. Compared to the hardware platform, the software is easier to promote and provides a cross-platform API (Application Programming

Interface). However, these kinds of tools are inefficient and poor in performance, so it is hard to satisfy the requirements of a future network. To be specific, take Geist [7] for example, which is a tool that generates realistic traffic for exercising web-servers and e-commerce front-end servers but the insufficient performance allows limited access. In a nutshell, common instrumentation usually provides limited flexibility and available software generators have poor performance. In this paper, we join both high-performance equipment and programmable software design.

In literature, a huge amount of work exists on the characterization, modeling and simulation of the network workload such as, live streaming media [8], and YouTube traffic [9]. Unfortunately, we cannot state the same for the generation of a realistic network workload based on those methods. Many other studies also have contributed in the field of Web traffic generation. Zinke [10] points out that the Web traffic generator has two requirements. A real workload is defined as a sequence of requests which are received from a real world web server. A representative workload is defined as a generated workload which has the same characteristics as a given real workload. When it comes to the realistic network workload, numerous researches have been done. In order to formulate different kinds of traffic, Cheng [11] proposes a HTTP traffic generator to generate user-defined HTTP traffic and analyze the performance under different network characteristics. Given the representative workload, based on the characteristics of the present representative workload, Botta [12] proposes a method for the realistic network workload to study the emerging networking scenarios with multidimensional heterogeneity and scale. However, most of the related work cannot generate a realistic network workload over a long and large time scale.

In this paper, aiming to design a Web traffic generator with the characteristics of verifying the future network architectures, we propose a flexible framework to support arbitrary models. Firstly, the system should generate realistic web traffic over a large timescale; secondly, a large number of users can certainly be described; finally, multiple operators can use the platform simultaneously. The generic Many-Core architectures with lots of cores [13], a remarkable new field, are providing new opportunities for a high-performance traffic generator. We develop a high-volume parallel design for a Web traffic generator on Many-Core processors, called TGMP. Unlike other traffic generators, we concentrate on the hierarchical architectural design, which allows for better control of the traffic and a more scalable generation. In order to generate a realistic network workload over a large time scale, we combine the method of user behavior with the user-control method by cubic spline interpolation.

We first explore how the TGMP can be leveraged to efficiently overcome the limitations in current Web traffic generator architectures with low flexibility, low scalability and high cost. After that, to break this stalemate, we have tackled two main technical issues. First, we perform large-time-scale flow simulation. Specifically, we use cubic spline interpolation to handle the traffic self-similarity in one hour or one day, which keeps the corresponding traffic flow that reflects the real network conditions. Compared to solutions performed by traffic replay, our solution does not need to capture the packet in advance. Second, we explore new parallel opportunities provided by Many-Core processors. More specifically, we propose a parallel design for implementing TGMP on the TILERAGX36. Our solution is inspired by Jiang [14], who designs a NIDS (Network Intrusion Detection System) on such processors on which tasks are split and sub-tasks are allocated to run concurrently through decomposition techniques. The contribution of our work can be listed as follows:

1. The software architecture of a Web traffic generation system is proposed by using the hierarchical design methodology including the control layer, virtual user layer and traffic generation layer, to provide high scalability.
2. We present a user-control method using the cubic spline interpolation based on the analysis of the Web user behavior simulation method of accessing the real server. The method enables the system to generate the background traffic with the characteristics of a real network over a long time scale according to the Internet user's access time in different scenarios.

3. In order to meet the requirements of concurrency, we have implemented a TGMP prototype. Three high concurrency strategies are employed, which enable the system to simulate a large number of virtual users at the same time, and generate more Web traffic.
4. We has been implemented and deployed in the real network at the third floor of the YiFu building in the CQUPT campus. The experiments show that compared with other systems, TGMP yields a more satisfactory performance in which 50,000 users access the Web server simultaneously.

The structure of this paper is as follows. The related Web traffic generation systems and the challenges in developing TGMP are introduced in Section 2. Section 3 presents a layered architecture of TGMP. Section 4 describes the cubic spline interpolation algorithm. Section 5 focuses on the high concurrency strategy. Section 6 presents the experimental results. Finally, we conclude this work in Section 7.

2. Background and Challenges

In this section, we discuss the advantages and the disadvantages of the currently existing traffic generations. The purpose of the discussion is to make a wise decision in the design of TGMP.

2.1. Overview of the TILEGX36 Architecture

Many-core processors, which are usually equipped with a larger number of cores per processor (tens to hundreds of cores per processor compared to up to six or eight in existing commodity multi-core processors), have emerged in the past years. In particular, each core in these processors can run a full operating system, which provides high flexibility and simplicity for software development.

The TILEGX36 processor is a typical many-core processor with 36 homogeneous cores, which are organized in a 6 × 6 grid, interconnected using an on-chip network structure as shown in Figure 1. Each tile is a full-featured computing system that can run independently. The (User Dynamic Network) UDN is executed in processes or threads, which effectively reduces the power consumption and communication delay.

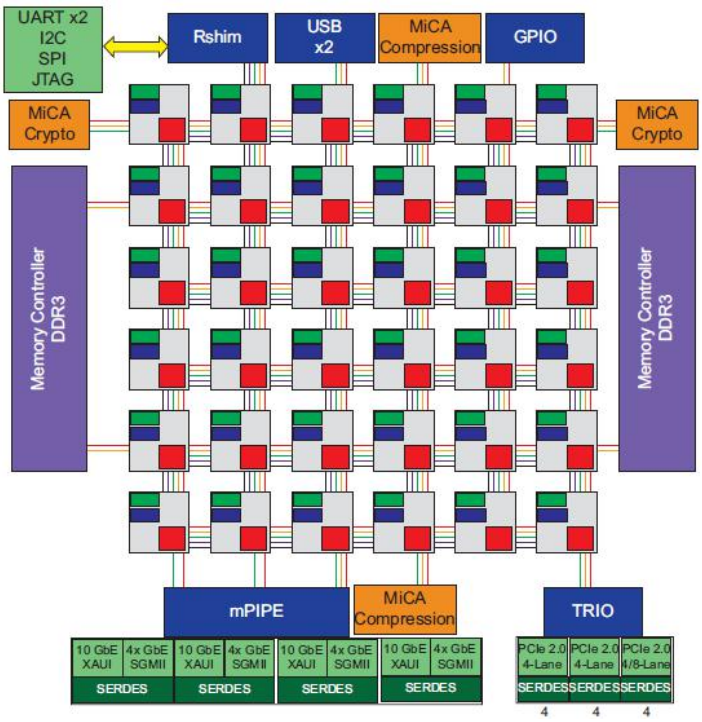


Figure 1. Overview of the TILEGX36 architecture.

2.2. The Existing Approaches of Traffic Generation

Accurate modeling and generation of a realistic network workload are difficult and challenging tasks because of heterogeneity, scale and complexity of the current Internet. In literature, a substantial number of works focus on the modeling and simulation of traffic generation. At present, the traffic generation has the following methods:

1. Traffic Replay [15,16]

Traffic replay tools are used to repeat real traffic scenarios. Traditionally, such a tool relies on software solutions that capture the whole traffic trace, and send the trace to the test network. Some special hardware devices should be provided. For instance, Qiao [15] realizes a traffic capture and replay system based on The Add-on Card, which interprets the detailed logic design of UDP pipeline on FPGA (Field-Programmable Gate Array). Although the Web traffic generator based on hardware implementation can produce high-volume test traffic, its poor scalability makes it difficult to integrate with the experimental network. GoReplay offers us the similar idea of reusing our existing traffic for testing, which makes it incredibly powerful. Nonetheless, these kinds of approaches can only reflect a period of time instead of a long and large time scale.

2. Traffic Model [17]

Some existing tools, such as WebStone [4], Web Polygraph [18], and httperf [19] are based on synthetic models of Web traffic. The network traffic model should be built by the study of the characteristics of a traffic generator. For example, Xu [17] set each virtual user with a corresponding configure file, and these files determine the visiting paths, visiting moments and stay time of virtual users based on the Continuous Time Markov Chain. These models are developed analytically and then validated experimentally with measurement studies. Although it allows fine-grained control over behavioral aspects, some serious drawbacks are also inevitable. For instance, it can only reflect the macroscopic characteristics of the network traffic. In particular, such tools can successfully create realistic traffic mixes as a function of overall load. However, these tools typically cannot provide good performance.

3. User Behavior [20,21]

The process of visiting the Web site can be roughly divided into the following three stages: First, a user starts a conversation by selecting a link what they are interested in; Second, after browsing, they often click another link, in which they are interested, in a relatively short time; Third, when he obtains the required information, he will stay for a long time with an inactive state. In particular, Figure 2 shows the important traffic parameters, including think-time, session-time, etc. In such a case, traffic characterization must be in line with the behavior of an individual user. In this paper, we first analyze web workload characteristics such as file sizes, mean think time, and the number of requests made to an individual file. We utilize the typical ON/OFF [20] internet accessing statistical model in which its key parameters are procured from real traffic analysis. Thus, it can generate realistic network traffic.

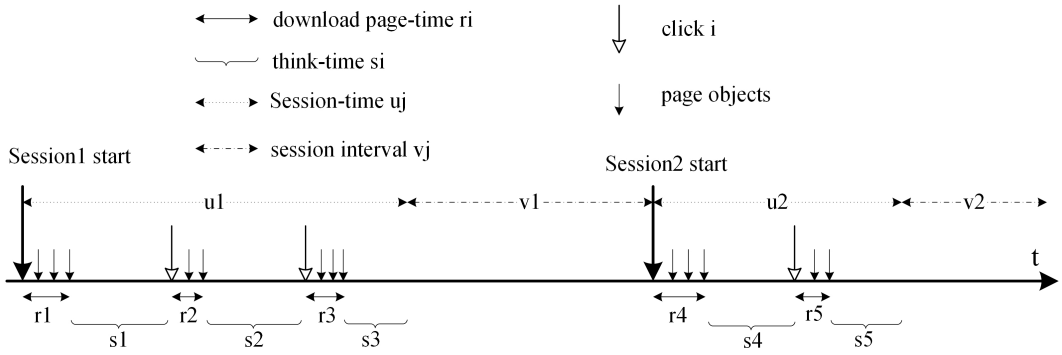


Figure 2. User browsing behavior.

2.3. Challenges

Generating high-volume, stable, realistic test traffic is crucial for assessing the performance of network devices in a reliable way and under different stress conditions. In this paper, we borrow from the hierarchical architecture design [22] in that it separates control plans and perform plans, and exposes the functions' interface to higher layers to deal with the dynamics of the network.

Two main technical challenges should be considered. First, its realistic replication is a challenging task over the large scale of time. The multidimensional heterogeneity of the current internet exacerbates the seriousness of the problem. However, the existing schemes can only reflect a period of time or the previous characteristics of the network. To address this issue, we need to generate the background traffic with the characteristics of a real network over a long and large time scale according to Internet users' access time data in different scenarios. Therefore, we present a method based on the cubic spline interpolation of the user control method to simulate the characteristics of a real network. Meanwhile, we generate a realistic network workload by accessing the real server.

Second, existing schedules have mostly been low-efficient due to the limited hardware resources and inefficient process. To better improve the scalability of the system and satisfy the required flow rate, the support for high-performance and configurable experiments is greatly needed in such context. Furthermore, in large-scale networks, tests have to be performed automatically because the size of the system under test may prevent the manual performance of activities on each and every host involved in the experiment. Even though existing traffic generators are quite useful, nonetheless, most of them suffer from the following: (1) It may need special equipment which is expensive or not commonly used. For example, MoonGen [23] is a flexible high-speed packet generator. A key feature is the measurement of latency with sub-microsecond precision and accuracy by using hardware timestamping capabilities of modern commodity NICs (Network Interface Cards); (2) It is not trivial to generate traffic data in arbitrary spatial regions using existing traffic generators. For example, DPDK (Data Plane Development Kit) is a set of libraries and drivers for fast packet processing, which is widely used in the field of traffic generators, such as MoonGen [23], TRex [24]. However, it can only capture the packet at the data link layer instead of dealing with it at the application layer. In order to generate real web traffic, the packet must enter the kernel protocol stack. The availability of generic many-core architectures with tens to hundreds of cores per processor which have the features of low-cost and feature-rich, is offering new opportunities for parallelization and extensibility. Taking advantage of the efficient network architecture , a schedule should be designed, which can efficiently address the challenges of high-performance and improve the efficiency to leverage the Many-Core processors.

3. Traffic Generator Systems Architecture

The development of a control system has attracted significant attention. Supporting software, which needs to be highly adaptable to changeable network scenarios, remains one of the greatest obstacles to a Web traffic generator. In this section, we will present the layered architecture of our

traffic generator and its main components. Figure 3 gives an overview of TGMP architecture and the layered system architecture mainly includes an application layer, a control layer, a virtual user layer, and a traffic generation layer. The traffic generation experiments can be achieved by the coherent cooperation of these layers; whenever a traffic experiment is initiated, by calling northbound APIs, the experiment parameters and system setups can be forwarded to the lower layers including control layer, virtual user layer and traffic generation layer. Also, using southbound APIs, the real-time feedback information of the traffic generation tasks would be transmitted from the lower layers, virtual layer and traffic generation layer. Each of the components will be presented in detail in the following sections.

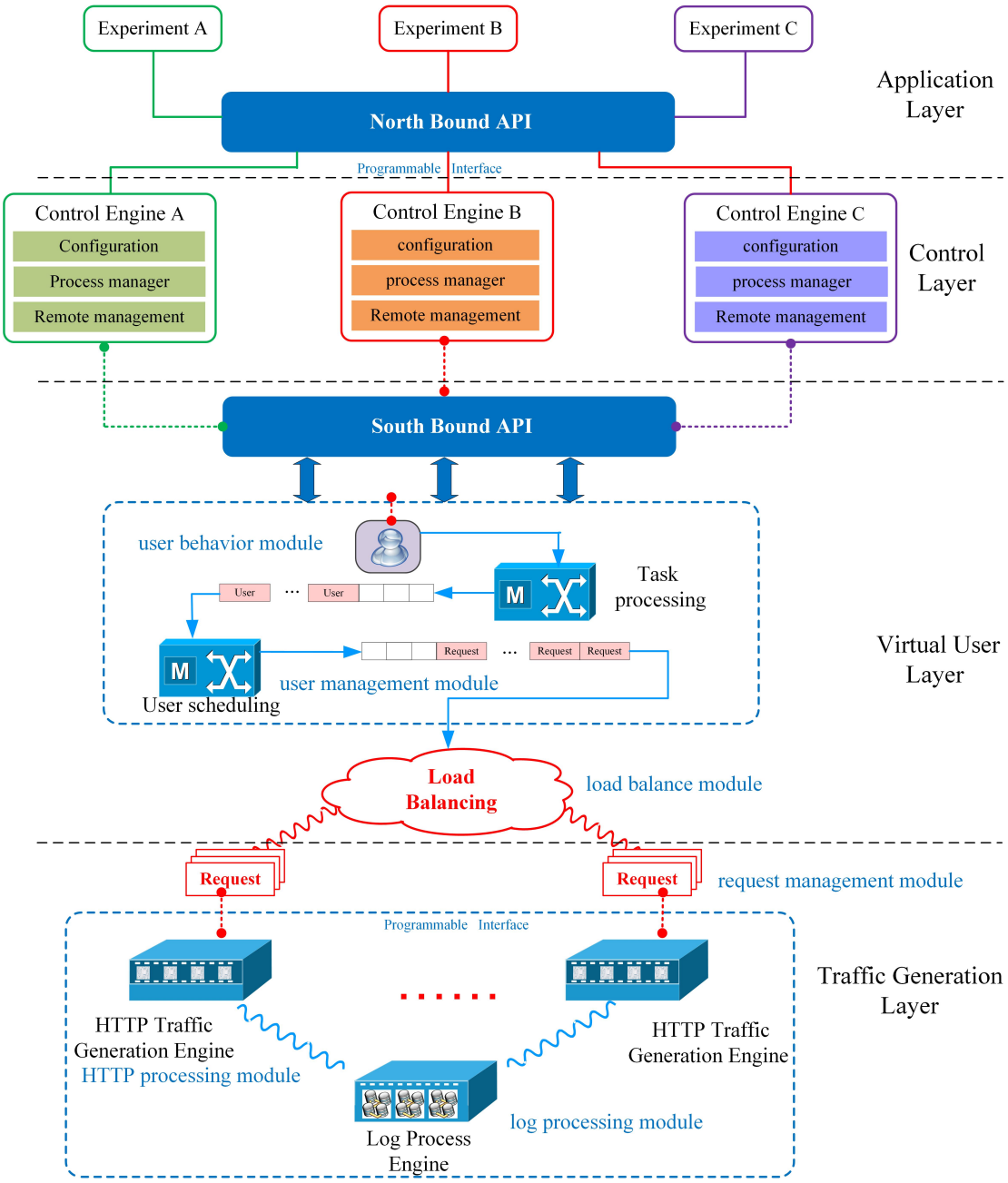


Figure 3. The layered system architecture of TGMP.

(1) System Components

Application layer contains abundant user-customized traffic generation applications which are designed as scheduled traffic generation experiments. North Bound application programming interfaces (APIs), provided by the control layer, are called by the application layer to enable its message exchange ability and system state monitoring function.

Control layer is assembled by the process management module, configuration module and remote management module. The process management module is responsible for process creation, multi-process parallelization settings, event-driven initialization and signal event registration. The configuration module provides the ability of setting the prerequisite parameters for the system to initiate unerringly. The remote management module monitors and handles the requests and messages from the application layer.

Virtual user layer provides the user management module, user behavior and load balance module. The user management module is responsible for virtual users' resource scheduling, allocation and isolation in each defined traffic generation task or experiment; the user behavior module is in charge of the complete implementation of the Web user behavior model, controlling each virtual user's web browsing actions; the load balancing module is to assign the amount of work that traffic generation has to do between two or more processes, so that more work can be done in the same amount of time.

Traffic generation layer can be divided into the request management module, HTTP processing module and log processing module. The management module is responsible for the requested object's creation; management of the resource pool; acquiring the target URL and HTTP request message parsing and structure; the establishment of a TCP connection; and network I/O event registration and callback processing. The HTTP processing module as the data processing module, mainly completes HTTP response message asynchronous parsing and discards the HTTP response entity. The log processing module will complete the process of user access log records and periodically access the log pushed to the log database.

(2) Use Cases

A use case is employed in order to have an overview of how the structure works. As the centralized management maintaining multiple connections for each experimenter, the web server provides the opportunity to achieve deployment and configuration, which allows many operators to use the Web Site at the same time. The simulation process for a Web traffic generator as shown in Figure 3, mainly contains the following steps:

In the first phase, the experimenter sends the simulated service message to the control layer, which includes the number of simulated users, the number of http sessions, interval time between sessions, the number and frequency of web clicks.

In the second phase, the control layer receives messages from the application layer, and then encapsulates it into a specific message format, which is sent to the virtual user layer. What is more, it keeps active until the processing result arrives from virtual user layer.

In the third phase, the virtual user layer parses the messages which are sent by the control layer and assigned to the corresponding resource (such as the virtual user resource). Each virtual user is activated and the corresponding timed events are registered. Notably, when a virtual user's timer is triggered, its callback function is called to send a request message to the traffic generation layer.

Last, the traffic generation layer obtains the specified URL according to the request message, then establishes a connection with the target Web server. When an event is triggered, a readable or writeable I/O event is assigned. In this case, data interaction can easily be manipulated by many event mechanisms. Finally, the request process is pushed into the database in a certain format.

4. User Control Method by Cubic Spline Interpolation

This system requires the generation of realistic Internet traffic from a test scenario's perspective without having to emulate network components or protocols. The method based on user behavior

supports self-similarity by making each user equivalent in an on–off process. However, in the larger time scales (e.g., one hour or one day), the change is relatively stable. Based on the statistics of online time distribution by Baidu statistical traffic institute [25], this experiment covers more than 1.5 million sites. Figure 4 shows the number of Internet users in a single day, which obviously demonstrates a clear time of day diurnal pattern. The main implementation problem here is how to have the clearest time of day diurnal pattern base on the characteristics of the aggregate traffic without having to observe any obvious changes with time of day effect.

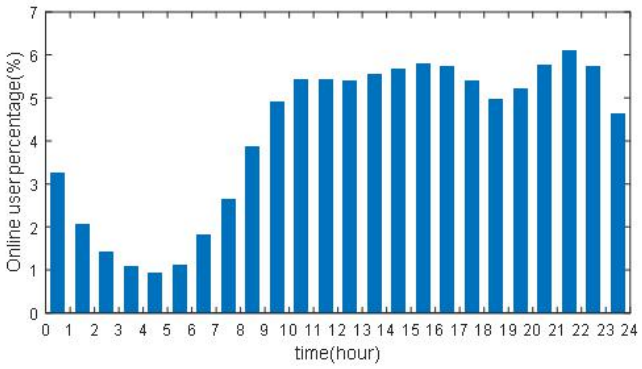


Figure 4. Internet users vs time of day.

In this section, we discuss the cubic spline interpolation algorithm in Table 1, which achieves the user time curves of refinement and keeps the change trend of users. We firstly finish the data pre-processing, then construct the cubic spline interpolation model, and subsequent sample refinement, then lastly carry out user control. More specifically, we assume the number of virtual users is Nv. According to the actual online user percentage in Figure 4, we obtain virtual online user percentage $V(x)$ in 24 time slots. In step 3, discrete $V(x)$ is interpolated in chronological order to obtain a continuous function $S(x)$ by the cubic spline interpolation algorithm. In steps 4–10, we take a more accurate sampling for $S(x)$ and obtain the virtual user reference set B. In steps 11–18, we continuously adjust the number of active virtual users in each time slot based on B. Finally, this algorithm achieves the user time curves of refinement and maintains the user's changing trend. In this way, we can obtain a more reliable load for Web traffic generation. We have built a prototype system to verify how our algorithm can generate traffic effectively in a field test.

Table 1. The numbers of users versus time curve, processed by Cubic Spline Interpolation.

Algorithm 1. Cubic Spline Interpolation Algorithm.

Input: the number of actual users N, actual online user percentage $T(x)$.
Output: the virtual user reference set B, the number of active virtual user A.

```
Compute the number of virtual users N_v which satisfies N_v ≥ N * max[T(x_i)]
where i = 0, 1, ··· , 23;
Obtain virtual users-time percent data V(x_i) in each time slot i calculated as
V(x_i) = N * T(x_i)/N_v, (i = 0, 1, ..., 23);
Use the cubic spline interpolation algorithm, and compute continuous time
function S(x) based on interpolation processing V(x);
Identify the time slot j which divides 24 h into 30 s;
Initialize j = 0;
while x_j ≤ 24 h do
        B[j] = S(x_j);
        j++;
end while
Obtain the virtual user reference set B in chronological order;
Assume the number of active virtual user A_j in time slot j;
while x_j ≤ 24 h do
    if A_j > B[j] * N_v then
        Delete A_j − B[j] * N_v out of active users.
    else
        Add B[j] * N_v − A_j extra active users.
    end if
end while
```

5. The High Concurrency Strategy

In this section, we explore the concurrency strategy of TGMP systems on Many-Core architectures [26]. The plan includes three strategies. We first modify the Linux kernel parameters to provide basic conditions for high concurrency. For the purpose of evaluating high-traffic, we have integrated the event-driven programming [27] into our proposed design. Furthermore, to take advantage of the particular feature of the TILERAGX36 hardware, we break the bottlenecks by using the task decomposition and task mapping on TILERAGX36 platforms to improve the parallel processing.

5.1. Optimization of Parameters

The default Linux kernel parameters are the most common scenario, which obviously does not support high concurrent processing of a traffic generation system. Therefore, some parameters should be reconsidered in terms of their size and usage. Three aspects are mainly be considered, including file descriptor, port numbers, and TCP parameters. Based on the capabilities of the operating system, the right parameters can be set manually.

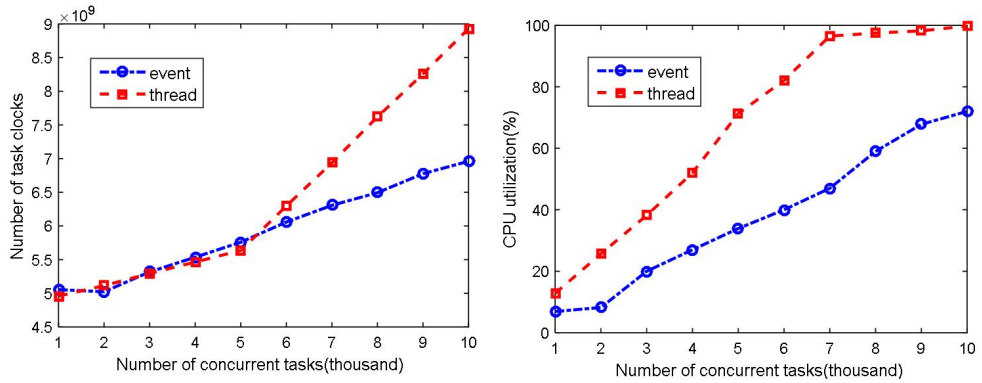
5.2. Event-Driven

Currently, the high concurrency strategy of the operating system follows two different design concepts: multi-threaded and event-driven [28]. Multi-threaded applications can be performed simultaneously on a single process by sharing the process resources, which makes it easy to communicate. The multi-threaded concept aims to increase the utilization of a single core by using thread-level as well as instruction-level parallelism. The event-driven concept, as a new method of programming, is not triggered by the sequence of events but is random, which is employed in many high performance open architectures, such as Nginx [29], and Memcached [30]. In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected. Motivated by this, in order to satisfy the special requirement of the high concurrency and avoid redundant processing, a wise decision must be made. In the following

paragraphs, to juxtapose the performance on the TILERAGX36 processor, an abstract application scenario is described.

First, a process is bound to the designated tile on the Tile-Gx36 processor. Second, we assume a task that achieves from 1 to n of the cumulative calculation, where n is selected from a group of random numbers ranging from 10,000 to 20,000. Third, event-driven and multi-threaded applications perform 10 times respectively. To characterize the processing performance of the two different strategies, the task execution time and the total number of CPU clock are measured.

Figure 5a shows that the task execution time keeps consistent when the task number is less than 5000. When it is higher than 5000, the multi-threaded approach increases significantly. When it comes to CPU utilization, it can be observed that the multi-threaded approach grows faster in Figure 5b. The event-driven application is processed in order, thus expensive context switching between tasks is not necessary. Motivated by these observations, the event-driven application is an effective method to adopt.



(a) Task execution Time (b) CPU utilization

Figure 5. Compare the event-driven and multi-threaded applications.

5.3. Parallel Architectures on TILERAGX36

At a high throughput level, more processes or threads may be required than that in the traditional platform in order to support the platform efficiently. This section describes how to design the parallel architectures on TILERAGX36 platforms. To reduce the contention of hardware resources, a reasonable allocation of tasks should be assigned in different cores. Since our goal is to reduce the energy consumption of processes generated mostly from inter-process communication (IPC), the IPC schemes will be discussed. Meanwhile, dynamic load balancing is a particularly important method to control the performance.

5.3.1. Task Decomposition

As described in Section 3, the system architecture is layered and centralized. Task decomposition can be considered as parallel processing of the basic strategy. It is used to solve complex computational problems by splitting them into sub-tasks quickly on a multiprocessor to achieve operation simultaneously. To demonstrate the processes of every layer in real-world settings, the implementation is shown in Figure 6. Blue, green and purple denote the control layer, virtual user layer and traffic generation layer, respectively.

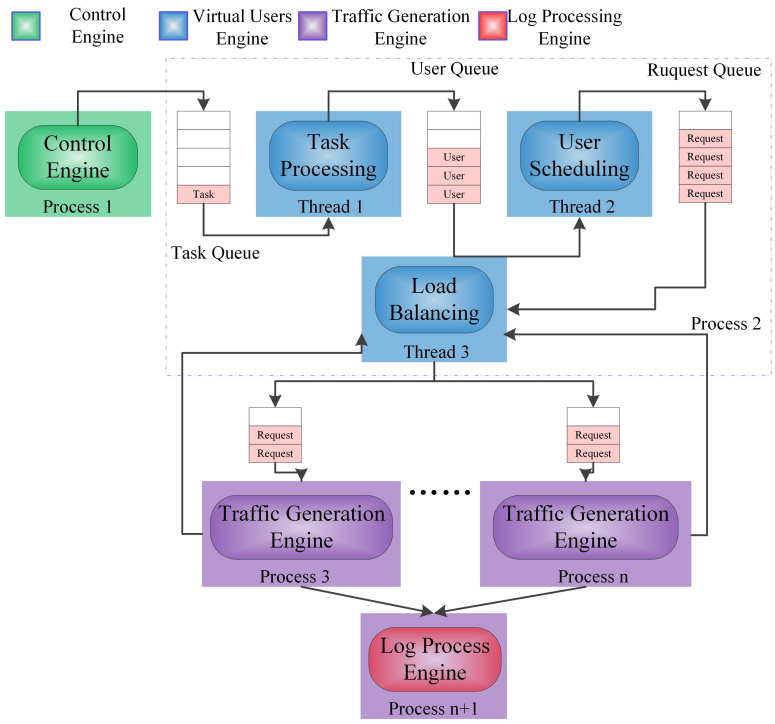


Figure 6. Parallel processing.

Taking full advantages of the TILERAGX's parallel processing ability, we use one control process as the father process to fork its child processes. The implementation method of parallel processing in this system, based on multiple processes, can be divided into the control process, virtual user processes, log processing and traffic generation process. It has been shown that a multi-threaded application has comparatively better event processing capabilities in terms of meeting processing deadlines than that of a multi-process application [28]. Task processing, user scheduling and load balancing all belong to the virtual users layer, so the multi-threaded application is utilized, with other modules executing a multi-process application. Since it is difficult to complete the assignment with a single engine, it is compulsory to move the traffic generation task to numerous engines.

According to the descriptions of use cases in Section 3, the first and second phase are relatively simple; we mainly describe virtual users layers and traffic generation in detail and explain the interaction process between them.

A. Virtual users layer

As a middle-level, the virtual users layer is mainly responsible for the management of virtual users, which needs both to handle the message that was sent by the control layer, and also deliver requests to the traffic generation layer. It is necessary to achieve flexible user management and resource allocation so as to make it available to multiple experimenters at the same time. According to the demand of experimenters, virtual users were divided into different users groups.

The user management module is responsible for implementing the experimenter's experiments while, at the same time, involving more virtual users of the resource allocation, user behavior parameter configuration and scheduling. In each experiment, the experimenter usually needs to create a large number of virtual users and the corresponding user's group, while these resources must be free at the end of the operation. The frequent memory operations can lead to a lot of overhead, and are easy to cause memory leaks and other issues. By establishing a resource pool and a set of connections, the use management strategy can be effective to avoid frequent resource creation and release overhead. This module designs user_pool, group_pool and epm_pool based on the given object pool technology implemented.

Task processing: During the simulation task message processing, some modules must be initialized according to the contents of the message. Firstly, we initialize the free queue (free_q) and active queue (live_q) , and then obtain the specified number of virtual users from user_pool and insert them into free_q; If the message contains scene parameters, the scene parameters are processed using the cubic spline interpolation algorithm described in Section 4.

User scheduling: User scheduling, which is responsible for scheduling a large number of users to achieve and generate the request message to join the request queue, can be divided into two sub-processes: user activation and user number control. If the task queue is not empty, the scheduler can scan the task queue for a certain period of time and activate the users in each group with a certain frequency to join live_q. If the user group contains scene parameters, it will be processed by the user control process when the number of active users reaches the number of scene start points. It controls the number of active users based on the user profile. Otherwise, all users in the group are activated, and the user behavior module controls the behavior of each user.

Load balancing: The numerous deployments of traffic generation engines offer the opportunity to exploit multiple accesses to the improvement of the concurrent performance. The load balancing module assigns the request queue messages to each traffic generation engine.

B. Traffic generation layer

Request management module: In order to ensure concurrency, asynchronous programming has to be used. In this module, the theory of finite-state machines is introduced. The description of the global state machine at this level is shown in Figure 7. The blue line represents the general state transition process, which shows that this state may takes several asynchronous processes to complete; the green line indicates that there is a pending URL transfer process; others denote the transition of the error condition during status processing. When a request message is received to indicate the start state, then a request object is fetched from the request pool and a page URL (a URL containing a plurality of embedded resources) accessed by this request is acquired. Finally, the result will be sent to the log processing module while the request object is reclaimed.

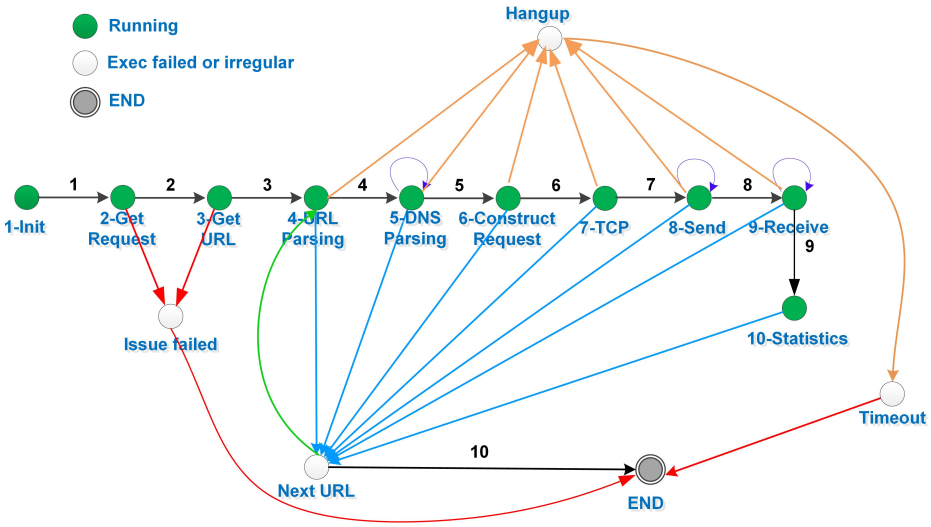


Figure 7. The global state transition.

HTTP processing module: We do not care about the specific content of the HTTP response, and there is no need to parse the content of the Web document. The HTTP processing module only parses the HTTP header to determine the length of the response entity, the coding and server information.

Log processing module: It is used to push the log into a database. When a new log is coming, using bulk inserts is a convenient way to improve efficiency. In addition, timing events can be used effectively to enhance concurrency.

5.3.2. Task Mapping

On the TILERAGX36 platform, by binding on the designated CPU rather than any CPU for a certain task or process that implements task mapping, each core in the processors can run a full operating system, which provides high flexibility and simplicity for software development. Under these circumstances, scheduling that process to execute on the same processor can improve its performance by reducing performance-degrading events. Additionally, it can effectively improve the cache hit ratio and reduce the number of memory accesses.

Furthermore, when a process or thread is bounded with one CPU, Linux kernal would not take it into the CPU schedule any more. As a result, the execution expense of the program would be largely reduced. For the Tile-Gx36 platform, the CPU affinity [31] is set by the following steps. Firstly, procuring the affinity set of the program; secondly, bounding the task process with a specific CPU according to its unique index in CPU affinity. The details can be described in Table 2.

Table 2. The details of the CPU affinity.

```
//procuring the CPU affinity
cpu_set_t cpus;
if (tmc_cpus_get_my_affinity(&cpus) != 0)
      tmc_task_die("tmc_cpus_get_my_affinity() failed.");
//bind to the allocated CPU
if (tmc_cpus_set_my_cpu(tmc_cpus_find_nth_cpu(&cpus, rank)) < 0)
      tmc_task_die("tmc_cpus_set_my_cpu() failed.");
```

Referring to the methods mentioned above, we can bound the decomposed tasks with multiple cores of Tile-Gx36. The control engine, virtual user engine and log engine are assigned with their own individual cores for processing. Each traffic generation engine is bounded with one core, thus the number of the traffic generation engine should be adjusted dynamically according to the restriction of the number of total cores. Moreover, Figure 8 describes the schematic diagram by specifying a CPU affinity setting for each process.

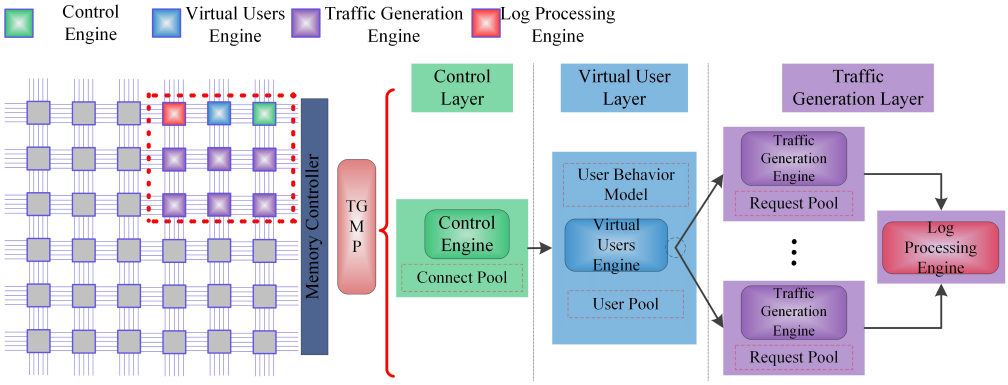


Figure 8. Parallel design on TILERAGX36.

5.3.3. IPC Scheme

Not only task decomposition and mapping process, but also the IPC scheme is assigned in Many-Core processors. Pipe, message queues, Unix socket, signal and shared memory are currently the most widely used genre of IPCs. The Tile-Gx36 processor provide a new method, the UDN, which is used to improve data transfers among tiles. However, the UDN is used to send small packets with the size of no more than 128 bytes. If the packet is too large or receives buffer overflow, the system will lead to a deadlock.

In our real-time processing tasks, each layer requires good two-way communication to ensure better supervision and process scheduling. The Unix socket exhibits a very low transmission latency and meanwhile supports full duplex mode, which provides a much better bi-direction communication. Figure 9 shows the communication between each engine, which consists of two parts. One-way communication is adopted in the traffic generation module, control module and log processing module, while others use dual-way communication.

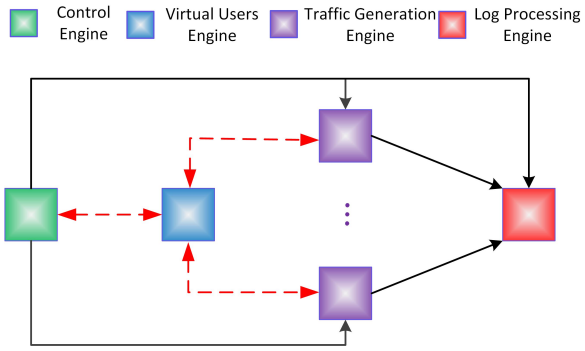


Figure 9. Based on Unix socket.

5.3.4. Dynamic Load-Balancing Based on the Minimum Number of Requests (DMR)

Due to the multiple traffic generation engines, dynamic load balancing is an important step to condition the parallel performance [32]. When the virtual users engine produces a mass of requests, it is necessary to predict which traffic generation engine to respond. To overcome this issue, very common approaches need to be raised based on the polling algorithm, weighting algorithm and hash algorithm to solve the load-balancing problem [33].

When the node performance of the traffic generation engine is basically the same, we can use a simple polling algorithm for load balancing. However, the traffic generation engine is based on the request object as a basic management unit, and the life cycle of each request object is not the same, leading to the evolution of the load being unpredictable. Furthermore, fast recovery of load-balancing can be very inefficient using a simpler approach, especially when it comes to a sudden increase in traffic or an abnormal process. In this paper, we present a dynamic load-balancing algorithm which accepts the dynamic change of the traffic generation engine based on the minimum number of requests (DMR). Each traffic generation process sends its own number of active requests to the virtual user layer in real-time. Then the load-balancing module updates the history record, and selects a traffic generation process with the smallest number of requests to send request messages. It can be classified as the weight class algorithm. The weight factor is the number of active requests in the process and the lower the weight, the higher the probability of selection.

6. Evaluation

In this section, we implemented and deployed TGMP in a real network to understand the performance of our system by conducting four groups of experiments, instances of its usage and directions for our future work.

6.1. Experimental Setup

TGMP has been implemented and deployed in a real network at the third floor of the YiFu building in the CQUPT campus. As shown in Figure 10, the deployment consists of Tile-Gx36, Nginx Web Service, LNMP (Linux + Nginx + Mysql + PHP) Web Server, etc. Nginx Web Service which has high concurrency performance was set up in many computers to conduct a comprehensive test on Web traffic. The Web Manage server provides a visual interface for the experiments. When the

experimenter sends the simulated service message to Tile-Gx36, the TGMP deals with the task, and then sends a request to Nginx Web Service to generate real traffic. Furthermore, all equipment is restricted to within the deployment in the local area network (LAN) due to the limited bandwidth in the experimental scene. We also deployed many random test experiments that perform different tasks. The TGMP is implemented in C and the Web management interface is realized in PHP. Notice that the design concept of the whole system is based on Nginx, which has focused on high performance, high concurrency and low memory usage. Additional features on top of the web server functionality, such as load balancing, caching, access and bandwidth control, and the ability to integrate efficiently with a variety of applications, have helped to make Nginx a good choice for modern architectures.

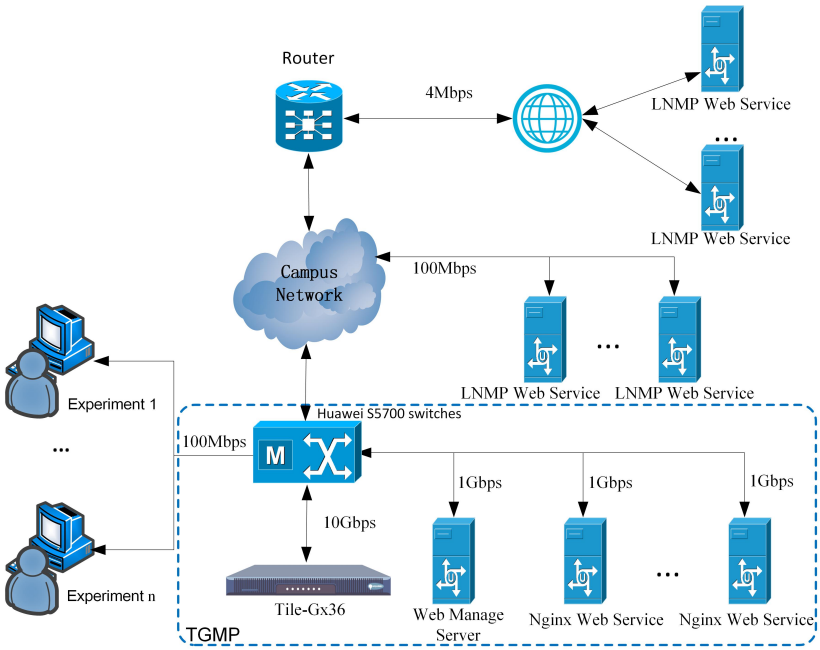


Figure 10. The deployment of the prototype system.

6.2. The Traffic Self-Similarity

Many studies [34,35] have reported on traffic characteristics with trends in self-similarity. This section presents whether the cubic spline interpolation of the user-control method has a large-scale of traffic self-similarity. Similar to [36], we also adopt the Hurst index which is the key indicator to evaluate the self-similarity of network traffic, whose length should be 0.71∼0.89 to judge the traffic self-similarity. Therefore, we select the stress test software, http_load, to compare with the network traffic based on user behavior. We construct the background traffic and grab the packets, and then compute the Hurst parameter to check the self-similar degree of network traffic by adopting the variance-time method [37] and the R/S method [38].

Figure 11 shows that the Hurst index of http_load is far from that of the theoretical value (0.71∼0.89). The green line means that the generated traffic self similarity and the red line stands for theoretical traffic self-similarity. Thus, the gap between these two lines can be used to measure the realistic characteristic of generated traffic. It means that the smaller the gap could be, the more realistic the generated traffic would be and the better the traffic generation system would be. In a comparison in Figure 12—the Hurst index is 0.87 and 0.83 respectively—the result indicates that the similarity improves. In this paper, the ON/OFF-based user behavior model is adopted, and multiple ON/OFF source overlays can generate traffic that is more consistent with the actual network [39]. The weak self-similarity of http-load traffic is mainly caused by the smaller size of the requested resource and the smaller frequency of requests.

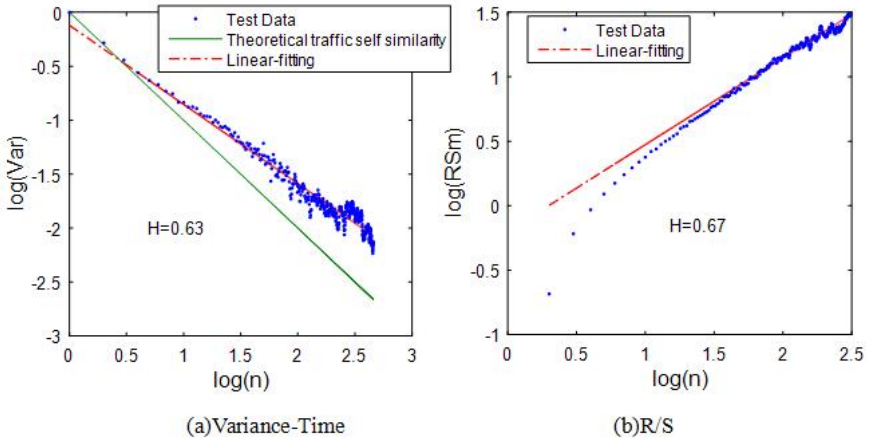


Figure 11. The Hurst index of http_load.

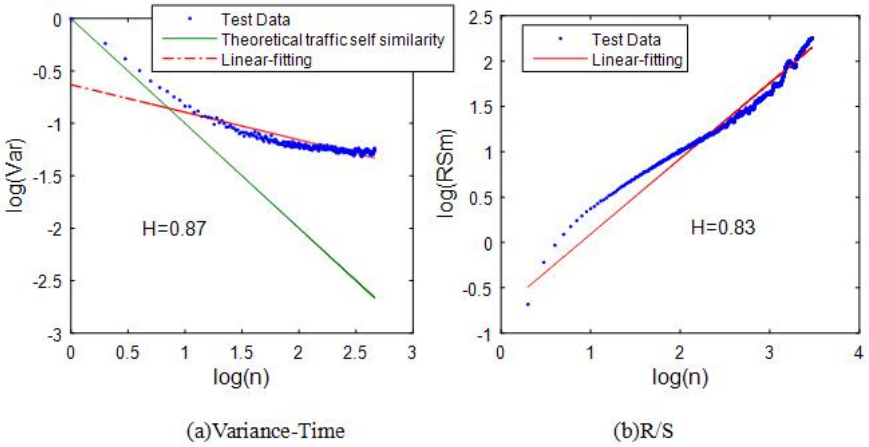
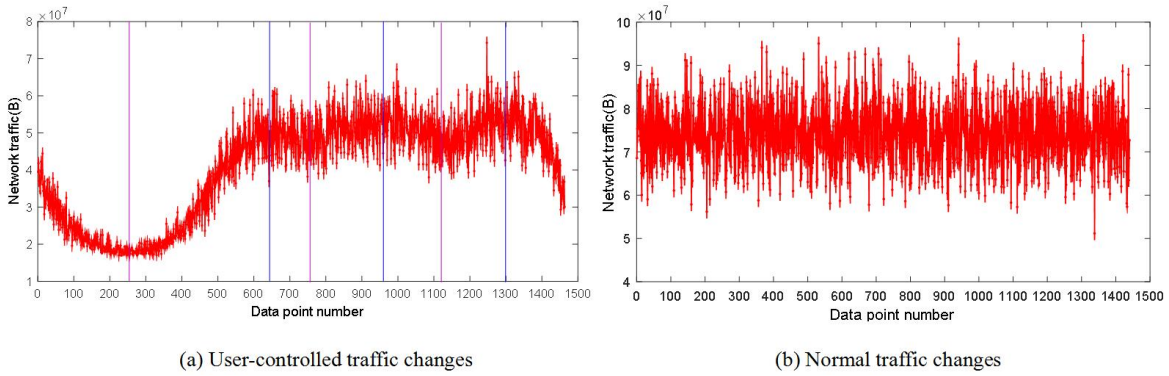


Figure 12. The Hurst index of TGMP.

To evaluate the effect on self-similarity based on the Cubic Spline Interpolation algorithm, we further study the realistic network workload over a long and large time scale. The experimental scene is shown to demonstrate the large-scale of flow simulation. To be specific, we assume a network of 100,000 users, and the executed User control is in accordance with Baidu statistics in February 2016 [25]. A shell script is executed per-minute in 24 h to collect the network traffic data. By contrast, we observed that there is a strong correlation in Figure 13a with the cubic spline interpolation algorithm. As expected, it can be seen that the system can control the number of virtual users effectively to achieve large-scale traffic generation which is more similar and in line with the actual network.



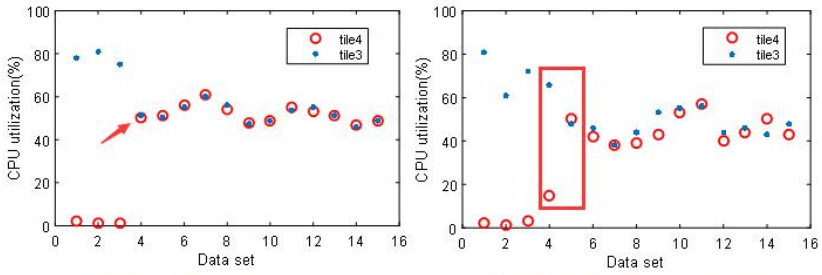
(a) User-controlled traffic changes (b) Normal traffic changes

Figure 13. The large-scale of flow simulation.

6.3. Load Balancing

The main goal of load-balancing schemes is to improve resource utilization efficiently and provide a high concurrency performance system. We start a traffic generation process and bind it to the tile 3 allowing the CPU usage to remain at about 80%. When the system is stable, we add another traffic generation process binding on tile 4, and then capture the two data of the respective CPU utilization every 0.001 s. In order to describe the change of the CPU utilization more clearly, the full data set is split across 1500, using every 100 data points, to calculate the average rate of the segment at a time. We have established the test scenarios which include DMR and Polling.

As can be seen from Figure 14a,b, when we add a new traffic generation process, the DMR algorithm achieves load synchronization for two processes in one cycle, while the Polling algorithm takes two cycles. This is mainly because the DMR can dynamically feedback the real-time access to each process of the real load situation, but Polling can only be completed with the request. In addition, the Polling has a greater variability than the DMR, which is mainly caused by the lifecycle of each request object. When a process is assigned to a larger lifecycle of the request message, it will take up more CPU resources, resulting in greater volatility. The DMR, however, can reduce the volatility by dynamic feedback function. Through the above two aspects of the comparative analysis, we can see that the proposed DMR has better performance.



(a) DMR algorithm of load balance (b) Polling algorithm of load balance

Figure 14. DMR vs Polling.

6.4. Concurrent Performance

In this experiment, the most important index, called concurrent performance, is evaluated, which can reflect the efficiency. The test mainly includes two aspects: first, we put a single traffic generation process into Tile-Gx36 and compare it with Nginx which has a high-level process architecture and can handle multiple connections within a single process under the same test scenarios. Second, we change the number of traffic generation processes and observe the phenomenon.

In order to ensure the consistency of the platform, we install Nginx in another Tile-Gx36 platform, and open a single work process model. Figure 15a shows the comparison of CPU utilization which remains basically the same. Thanks to the adoption of HTTP long connection, the change of the traffic and the number of connections along with the number of virtual users are in a simple linear model as reflected in Figure 15b,c. We observe that the change of the memory is small in Figure 15c. Furthermore, when the CPU utilization is around 80%, a single traffic generation process can support more than 7000 active virtual users, resulting in a traffic size of approximately 80 MB/s (640 Mb/s) and total of 2.3 GB.

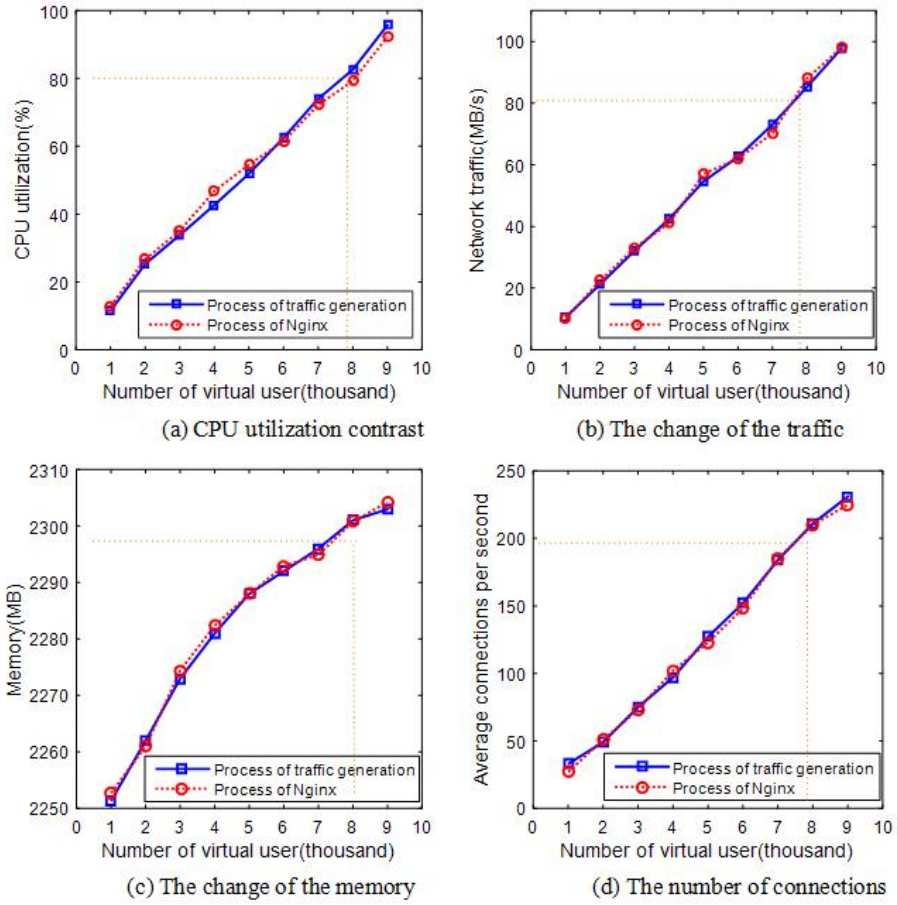


Figure 15. A single traffic generation process comparing the concurrency with Nginx.

The change of the number of virtual users is shown in Figure 16 when using many traffic generation processes. As observed in the previous experiment, all of those increase with the number of traffic generation process changes. Meanwhile, in Figure 16a, the number of virtual users has a strong but not a perfectly linear relationship, which may be caused by the response time. To be more specific, if the number of connections increases, mass traffic can lead to the response time of the server being lengthened in Figure 16d. In particular, when the number of traffic generation processes is more than eight, the response time of the server increases rapidly. As for increasing accessing users, the Web server processing time will increase dramatically. However, the CPU utilization is lower than we imagine. On the other hand, if there is a higher-performance Web server, the bottleneck will be greatly reduced. Above all, the traffic generation system has a good overall concurrency performance, which can simulate more than 50,000 users at the same time, and the flow rate is as high as 4 Gbps.

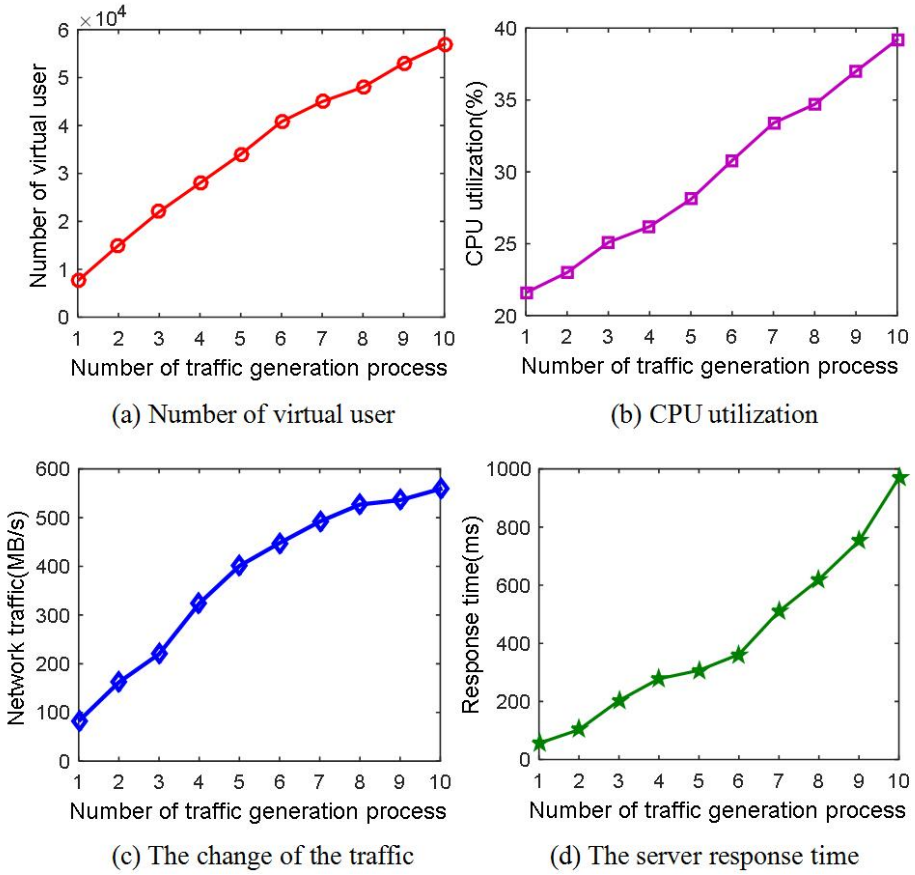


Figure 16. Many traffic generation processes.

6.5. System Stability

The stability test of the system is mainly to verify whether the system can run stably under the condition of a large load. The main specific test ideas are as follows: the system starts with opening 20 traffic generation processes and the number of analogue users is set to 5 million. At the same time, the test assignment continues to run for 24 h, so that all virtual users have been active. As shown in Figure 17a, the network traffic can be relatively stable. Figure 17b shows that the system's CPU utilization does not increase over time when fluctuating. The system which has a very slight volatility of memory can avoid a memory leak situation, as shown in Figure 17c. In summary, the system which has strong stability can produce long-term and stable flow.

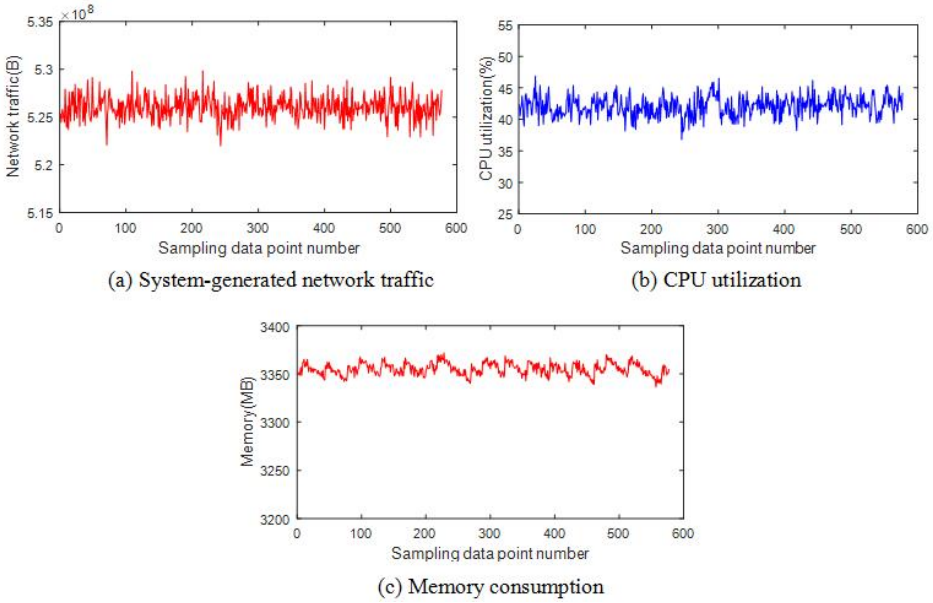


Figure 17. The stability of the system.

6.6. Comparison with Existing System

In order to understand the performance aspects of our traffic generator, we have compared it with some existing well-known tools in Table 3. Comparing with current proposed traffic generator systems, our parallel design on TILERAGX36 achieves a good performance for simulating 50,000 users accessing the Web server simultaneously. The obtained performance looks very promising. As it is possible to have up to two TILERAGX36 many-core processors on the same board, we can expect to almost double the attained performance in practice. Note that the main difference is the platform; MoonGen [23] and TRex [24] must rely on a special configuration. Moreover, the power consumption of the TILERA processor is 50 W at 1.2 Ghz which is much lower than others [14]. Another promising aspect of our design is that it adopts task decomposition and task mapping by binding on the designated CPU which enables our design to easily adapt and be extended to upgrades, especially as future processors will have an increasing number of cores.

Table 3. Compared with well-known tools.

Tools	Performance	Method	Scalability	Technology	Platform
TRex	200 Gb/s	Traffic Replay	Good	DPDK	Intel DPDK 1/10/40 Gbps interface support
TGMP	Simulate 50,000 users User Behavior	User Behavior	Good	Many-core processor	TILERAGX36
MoonGen	Up to 178.5 Mpps at 120 Gbit/s	Traffic Model	General	DPDK	Modern commodity NICs is support for multi-core CPUs

7. Conclusions

A traffic generator plays an indispensable role in the research of network architecture, new network protocol, network services, etc. The lack of scalability and efficiency in existing schedules motivates us to design an efficient general framework. The emerging Many-Core Processors, which are a revolutionary operation mechanism, can be leveraged to enhance performance. In this paper, we have presented the TGMP system, which is the first practical Web traffic generator operating on the TILERAGX36 processor. More specifically, a scalable, flexible and extensible layered system architecture, is designed for coping with highly changeable scenarios.

We adopt the design principles of cubic spline interpolation to generate a realistic network. In order to solve the problem of a representative workload, three high concurrency strategies are designed and TGMP is implemented in a real network to evaluate its efficiency. The experiment shows that TGMP can yield comparable efficiency with existing tools, but with much less cost and maintenance effort. Due to the limited space, this paper only considered the traffic of Web, therefore we will extend the system so that it is able to generate other types traffic, such as, video traffic, P2P traffic, etc. Furthermore, the idea of the design proposed in this paper could also be enlightening.

Supplementary Materials: The application is available at: https://github.com/FNRC/TGMP.

Acknowledgments: This research is supported by National Key Basic Research Development Program (973 Program) (2012CB315803, 2012CB315806) and Future Network Prospective Research Project of Future Network Innovation Institute in Jiang Su Province (BY2013095-2-03, BY2013095-5-07).

Author Contributions: All authors contributed equally to this work. X.W. and W.J. conceived and designed the experiments; J.W. and Q.W. performed the experiments; C.X. and G.Z. contributed analysis tools; X.W. wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Feknous, M.; Houdoin, T.; Guyader, B.L.; de Biasio, J.; Gravey, A.; Torrijos Gijón, J.A. Internet traffic analysis: A case study from two major European operators. In Proceedings of the International Symposium on Computers and Communications, Funchal, Madeira, Portugal, 23–26 June 2014; pp. 1–7.
2. Security Test Solution for App-Aware Devices. Available online: https://www.spirent.com/Products/Avalanche/ (accessed on 20 September 2016).
3. Measure the Quality of Experience of Real-Time, Business-Critical Applications With Converged Multiplay Service Emulations. Available online: https://www.ixiacom.com/products/ixload (accessed on 20 September 2016).
4. Thorndale, C.W. Webstone: The first generation in http server benchmarking. *Geophys. Prospect.* **1995**, *29*, 541–549.
5. Busari, M.; Williamson, C. ProWGen: A synthetic workload generation tool for simulation evaluation of web proxy caches. *Comput. Netw.* **2002**, *38*, 779–794.
6. Katsaros, K.V.; Xylomenos, G.; Polyzos, G.C. GlobeTraff: A traffic workload generator for the performance evaluation of future Internet architectures. In Proceedings of the 2012 5th International Conference on New Technologies, Mobility and Security (NTMS), Istanbul, Turkey, 7–10 May 2012; pp. 1–5.
7. Kant, K.; Tewari, V.; Iyer, R.K. Geist: A generator for e-commerce and internet server traffic. In Proceedings of the 2001 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Tucson, AZ, USA, 4–6 November 2001; pp. 49–56.
8. Veloso, E.; Almeida, V.; Meira, W. A hierarchical characterization of a live streaming media workload. In Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment, Marseille, France, 6–8 November 2002; pp. 117–130.
9. Abhari, A.; Soraya, M.; Meira, W. Workload generation for YouTube. *Multimed. Tools Appl.* **2010**, *46*, 91.
10. Zinke, J.; Habenschuss, J.; Schnor, B. Servload: Generating representative workloads for web server benchmarking. In Proceedings of the 2012 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Genoa, Italy, 8–11 July 2012; pp. 1–8.
11. Cheng, Y.; Çetinkaya, E.K.; Sterbenz, J.P.G. Transactional traffic generator implementation in ns-3. In Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques, Cannes, France, 5–7 March 2013; pp. 182–189.
12. Botta, A.; Dainotti, A.; Pescapé, A. A tool for the generation of realistic network workload for emerging networking scenarios. *Comput. Netw.* **2012**, *56*, 3531–3547.
13. Nedjah, N.; Calazan, R.M.; de Macedo Mourelle, L.; Wang, C. Parallel Implementations of the Cooperative Particle Swarm Optimization on Many-Core and Multi-core Architectures. *Int. J. Parallel Program.* **2016**, *44*, 1173–1199.

14. Jiang, H.; Zhang, G.; Xie, G.; Salamatian, K.; Mathy, L. Scalable high-performance parallel design for network intrusion detection systems on Many-Core processors. In Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Jose, CA, USA, 21–22 October 2013; pp. 137–146.
15. Qiao, S.; Xu, C.; Xie, L.; Yang, J.; Hu, C.; Guan, X.; Zou, J. Network recorder and player: FPGA-based network traffic capture and replay. In Proceedings of the 2014 International Conference on Field-Programmable Technology (FPT), Shanghai, China, 10–12 December 2014; pp. 342–345.
16. Huang, C.Y.; Lin, Y.D.; Liao, P.Y.; Lai, Y.C. Stateful traffic replay for web application proxies. *Secur. Commun. Netw.* **2015**, *8*, 970–981.
17. Xu, L.; Zhang, W.; Chen, L. Modeling Users' Visiting Behaviors for Web Load Testing by Continuous Time Markov Chain. In Proceedings of the 2010 Seventh Web Information Systems and Applications Conference (WISA), Huhehot, China, 20–22 August 2010; pp. 59–64.
18. Web Polygraph. Available online: http://web-polygraph.org (accessed on 20 November 2016).
19. Mosberger, D.; Jin, T. Httperf: A tool for measuring web server performance. In Proceedings of the First Workshop on Internet Server Performance (WISP), Madison, WI, USA, 23 June 1998; ACM: New York, NY, USA, 1998; Volume 26, pp. 31–37.
20. Varet, A.; Larrieu, N. How to generate realistic network traffic. In Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference (COMPSAC), Vasteras, Sweden, 21–25 July 2014; pp. 299–304.
21. Pries, R.; Magyari, Z.; Tran-Gia, P. An HTTP web traffic model based on the top one million visited web pages. In Proceedings of the 2012 8th EURO-NGI Conference on Next Generation Internet (NGI), Karlskrona, Sweden, 25–27 June 2012; pp. 33–139.
22. Howell, F.W.; Williams, R.; Ibbett, R.N. Hierarchical Architecture Design and Simulation Environment. In Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Durhan, NC, USA, 31 January–2 February 1994; Volume 94, pp. 970–981.
23. Emmerich, P.; Gallenmüller, S.; Raumer, D.; Wohlfart, R.; Carle, G. MoonGen: A Scriptable High-Speed Packet Generator. In Proceedings of the 2015 ACM Conference on Internet Measurement Conference, Tokyo, Japan, 28–30 October 2015; pp. 275–287.
24. TRex realistic traffic generator. Available online: https://trex-tgn.cisco.com/ (accessed on 25 November 2016).
25. Baidu Statistics. Available online: http://tongji.baidu.com/data/hour (accessed on 21 September 2016).
26. Wentzlaff, D.; Griffin, P.; Hoffmann, H. On-chip interconnection architecture of the tile processor. *IEEE Micro* **2007**, *27*, 15–31.
27. Reimers, S.O. Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems. In Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, Boulder, CO, USA, 31 October–3 November 2006.
28. Duffy, C.; Roedig, U.; Herbert, J.; Sreenan, C. An experimental comparison of event driven and multi-threaded sensor node operating systems. In Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops, White Plains, NY, USA, 19–23 March 2007; pp. 267–271.
29. Shao, Q.F.; Yang, T.C.; Hou, W. The design of high available single sign-on server of Nginx-based. *Appl. Mech. Mater.* **2013**, *241*, 2411–2416.
30. Litz, H.; Braun, B.; Cheriton, D. EXCITE-VM: Extending the Virtual Memory System to Support Snapshot Isolation Transactions. In Proceedings of the 2016 International Conference on Parallel Architectures and Compilation ACM, Haifa, Israel, 11–15 September 2016; pp. 401–412.
31. Tripakis, S.; Limaye, R.; Ravindran, K.; Wang, G.; Andrade, H.; Ghosal, A. Tokens vs. Signals: On Conformance between Formal Models of Dataflow and Hardware. *J. Signal Process. Syst.* **2016**, *85*, 23–43.
32. Vuchener, C.; Esnard, A. Dynamic load-balancing with variable number of processors based on graph repartitioning. In Proceedings of the 2012 19th International Conference on High Performance Computing (HiPC), Pune, India, 18–22 December 2012; pp. 1–9.
33. Sundaramoorthy, K. Adaptive Load-Balancing Algorithm with Rainbow Mechanism to Avoid Connectivity Holes in Wireless Sensor Networks. *Middle-East J. Sci. Res.* **2015**, *23*, 974–980.
34. Kourdis, P.D.; Bellan, J. Highly Reduced Species Mechanisms for iso-Cetane Using the Local Self-Similarity Tabulation Method. *Int. J. Chem. Kinet.* **2015**, *48*, 974–980.

35. Zhu, F.; Ding, M.; Zhang, X. Self-similarity inspired local descriptor for non-rigid multi-modal image registration. *Inf. Sci.* **2016**, *372*, 16–31.
36. Chen, J.; Tan, X.; Jia, Z. Performance analysis of seven estimate algorithms about the Hurst coefficient. *J. Comput. Appl.* **2006**, *4*, 059.
37. Zhang, H.F.; Shu, Y.T.; Yang, O. Estimation of Hurst parameter by variance-time plots. In Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, Victoria, BC, Canada, 20–22 August 1997; Volume 2, pp. 883–886.
38. Fu, L.Y.; Wang, R.C.; Wang, H.Y.; Ren, X.Y. Implementation and Application of Computing Self-Similar Parameter by R/S Approach to Analyze Network Traffic. *J. Nanjing Univ. Aeronaut. Astronaut.* **2007**, *3*, 016.
39. Pruthi, P.; Erramilli, A. Heavy-tailed on/off source behavior and self-similar traffic. *Communications* **1995**, *1*, 445–450.