THE UNIVERSITY OF SYDNEY

MASTER THESIS

Task Scheduling Algorithms for IoT

Author:
Shujian ZHOU

Supervisor:
Dr. Wei LI

A thesis submitted in fulfillment of the requirements for the degree of Master of Information Technology

in the

School of Information Technology

June 17, 2016

Declaration of Authorship

I, Shujian ZHOU, declare that this thesis titled, "Task Scheduling Algorithms for IoT" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

THE UNIVERSITY OF SYDNEY

Abstract

Faculty of Engineering and Information Technologies

School of Information Technology

Master of Information Technology

Task Scheduling Algorithms for IoT

by Shujian ZHOU

In recent years, Cloud computing has become an emerging technology due to its powerful and heterogeneous computing ability. Additionally, since it is a pay-as-you-go scheme, only the actual resources consumed would be counted. At the same time, Fog computing is recently proposed as a computing paradigm which is deployed locally, and it also provides real-time and low latency response service. Besides, IoT(Internet of Things) is sketching a vision in which all things (including physical devices, vehicles and other items), can convey information to other things and collaborate with them. In this thesis, to fulfill the requirements of computing tasks, for instance, short latency and efficiency, we have constructed a model of IoT that offloads its computing tasks to cloud devices and fog devices. Furthermore, to accomplish these goals, we have investigated different types of task scheduling algorithms and resource allocation policies. And a system model was constructed to implement these method. Therefore, these methods could be evaluated and compared. Lastly, the experiment result reveals that the improved scheduling policy outweighs the original one on both load balancing and scheduling length. Also, in practical use, the improvements will also benefit energy saving.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background of IoT

IoT has been one of the most promising paradigms which is supposed to build the connection among things, as to enable them to collect and exchange data. In this way, *Things* would be brought to a computer-based system, thus they can facilitate computing resources to achieve more functions. Furthermore, objects can collaborate to complete tasks, and also be used to collect data. For example, wearable devices, which work collaboratively with smartphones, has gained its popularity in recent years and been proved to be a successful application of IoT. Besides, more applications like smart grid and smart home appliance are seeing a significant growth in the market. On the other hand, it is a fact that *Things* devices usually have poor computing power, as they are sensors or embedded system inside various staffs including cars and home appliance. When heavy tasks were requested and expected be responded instantly, things could not finish them by themselves in expected time. Moreover, sometimes it may have significant impact on safety. For instance, when it is applied in a self-driving car, accidents may occur if the system cannot respond instantly to changes. Thus, with the limitation of Things devices, other methods should be involved with more computing power.

1.2 Inter-play of Cloud Computing and Fog Computing

In recent years, Cloud computing has been a emerging technology as it provides powerful and heterogeneous computing ability. A cloud server consists of a large amount of processors to work in parallel and collaboratively, so it can provide a powerful computing capability. Besides, as a large scale system, a Cloud server also contains diverse computing units including traditional CPU, GPGPU(General Purpose Graphic Processing Unit) and other processors for specific use, as to accelerate the processing of some specific tasks that are frequently requested. Additionally, in terms of cost, as it is a pay-as-you-go scheme, consumers will only pay for the service that they actually use, instead of purchasing devices. On the other hand, building a cloud computing centre consumes a massive amount of money, so there are only several ones distributed all over the world. Therefore, the connection delay to a cloud computing centre would varies in a wide range. Furthermore, clients access to cloud computing resources via the Internet, which is a complex network and contains diverse hosts. In this condition, network contentions may frequently arise, and there may be a significant number of hops from a client to a cloud server. Hence, it may consumes much time to access cloud service.

In order to alleviate the latency of cloud computing and provide better QoS, the revolutionary concept of Fog computing was proposed. Fog computing is defined as a distributed computing infrastructure that is applied to provide computing service to Internet-connected devices. The principle behind this paradigm is *edge-computing* [1], in which the devices are deployed at the edge of the Internet, including routers, gateways and access points. Thus, Fog devices can respond to requests instantly as they are closer to Things, while Cloud servers can utilize their powerful computing ability to process heavy tasks. In this way, the inter-play of Cloud computing and fog computing in the context of IoT will be of key importance as this model can ensure both response time and latency.

1.3 Task Scheduling Algorithms for IoT

As defined in [2], scheduling is a process in which works are assigned to resources to be completed. Therefore, the task scheduling algorithm is also important to achieve the goal of reducing latency and response time, since a effective scheduling policy could minimize response time, latency and maximize fairness [3]. Furthermore, it is also important to safety. Overall, the task scheduling policy has a significant impact on QoS(Quality of Service). Although task scheduling algorithms has been intensively studied in the subject of parallel computing and distributed computing, it is poorly investigated in the subject of IoT.

In this project, we construct an system model that consists of three levels, including IoT, Fog and Cloud, and applications will be processed on such architecture. After this, we apply a new scheduling policies and the traditional method onto this system and then present an evaluation of the final results.

1.4 Organization of This Report

This thesis report is organized as follows. Section 2 describes the related works done so far on this subject, including static scheduling algorithms and dynamic scheduling algorithms. After this, the design of system model and simulation will be introduced specifically in section 3. Section 4 presents the result of experiment and gives analysis on it. Lastly, this project is concluded in section 5.

Chapter 2

Literature Review

Task scheduling algorithms have been extensively studied, and it has been categorized in static scheduling and dynamic scheduling. The first section in this chapter will give a reveal of the static one, to briefly introduce the mechanism behind this method and its features. And dynamic scheduling will be introduced in the second section, including its specifications. The last section presents a reveal of related studies of Fog computing, Cloud computing.

2.1 Static Scheduling

Essentially, task scheduling problems is divided in two steps, assigning tasks to suitable computing resources and ordering the task executions on each of them. In this context, the characteristics including task computation cost, data size of transfer, dependencies among tasks and specifications of computing units are predefined, thus it is defined as a *static* model.

Since task scheduling is important to performance, it has been studied extensively and various algorithms are suggested in the literature. To be more specifically, these proposed algorithms are categorized in Heuristic Based approach and Guided Random Search approach. Furthermore, the Heuristic one is divided in following types,

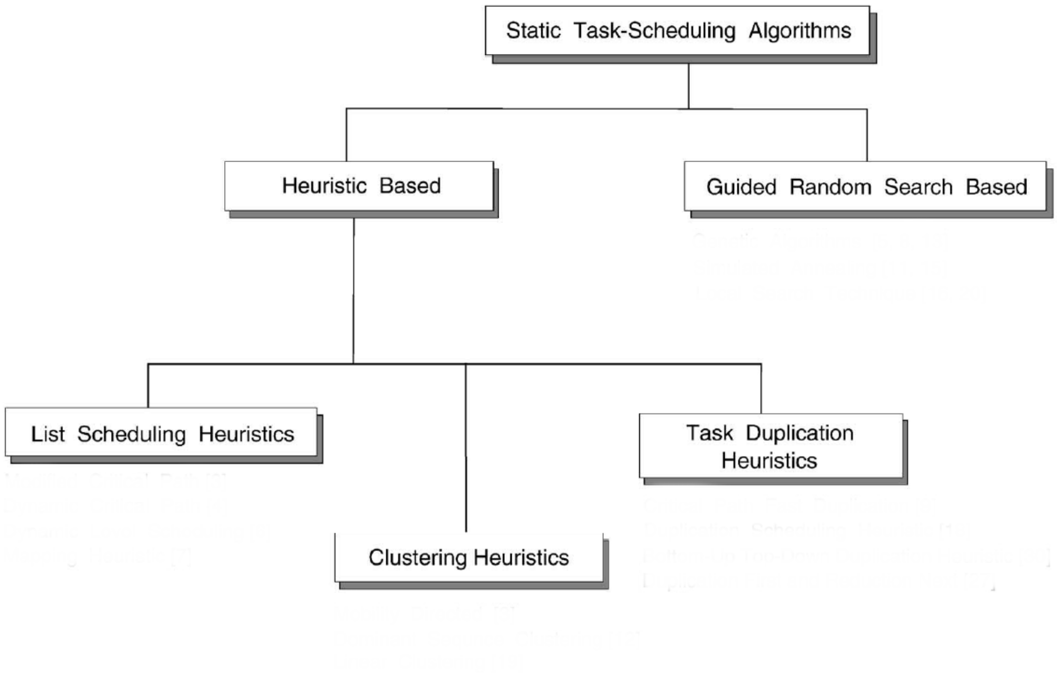
Static Task-Scheduling Algorithms
Heuristic Based
Guided Random Search Based
List Scheduling Heuristics
Clustering Heuristics
Task Duplication Heuristics

FIGURE 2.1: Classification of Static Scheduling Algorithms

List Scheduling, Clustering and Task Duplication. In this section, we will have a reveal of the Heuristic approach, and each type of this will be introduced in detail.

2.1.1 System Model of Task Scheduling

As for the model of task scheduling, it consists of the following three parts, an application, performance metrics and a set of computing units. Firstly, tasks of an application and the dependencies among them can be presented by a directed acyclic graph, $G = (V, E)$, where V is a set of vertices and E is a set of edges. In this graph, each vertex v indicates a task, and each edge e indicates inter-task communications. Besides, each vertex is marked with its computation cost, while each edge is marked with the communication cost. More specifically, for each edge $(i, j) \in E$, the precedence constraint is presented that task t_j can only start after the completion of task t_i. Apart from tasks, data is presented as a $v \times v$ matrix, where each element $data_{i,j}$ is the amount of data transmission between $task_i$ and $task_j$.

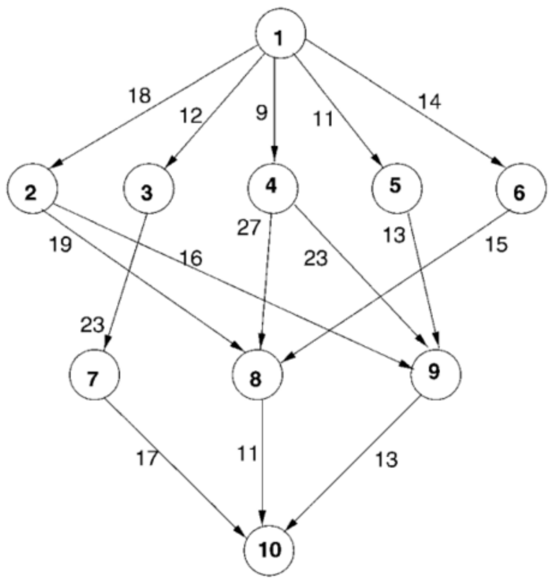

FIGURE 2.2: A sample task graph in DAG

Secondly, in terms of computing units, they are denoted in a set Q which consists of a number of processors. And in this environment, all computing resources are connected in fully connected topology, in which there is a dedicated connection between each two computing units, thus no contention will arise. Once Q was obtained and specifications of computing resources were acquired, the system can get the time spent on communication and computation for each task and on each processor. W is defined as the time consumption matrix, of which each element $w_{i,k}$ denotes the time consumption for task t_i on processor p_k. Next, as for communication cost, a $q \times q$ size matrix B will be used to present network bandwidth between each two nodes, and the communication startup cost of each computing unit is stored in a vector L.

Having acquired above information, before scheduling, the system will calculate for heuristics to help determine resource allocation and execution order. At first, each task t_i will be marked with its average computation cost $\overline{w_i}$ which defined as

$$\overline{w_i} = \sum_{k=1}^{q} \frac{w_{i,k}}{q}$$

And then, as we have the matrix B and vector L, the overall communication cost can be calculated by following equation, which is for data transferring from task t_i over processor p_m to task t_j over processor p_n

$$c_{i,j} = L_m + \frac{data_{i,j}}{B_{m,n}}$$

After this, we use the average value to be heuristic defined as

$$\overline{c_{i,k}} = \overline{L} + \frac{data_{i,k}}{\overline{B_{m,n}}}$$

Lastly, two attributes are defined as $EST(t_i, p_j)$ and $EFT(t_i, p_j)$, which signify earliest start time and earliest finish time respectively. These two attributes are involved to measure the performance of a partial schedule.

And then, when an application arrives with a set of tasks, the objective is to map these tasks to computing resources and set the execution order within each computing unit. Finally, task scheduling is expected to minimize the completion time.

2.1.2 List Scheduling

As its name suggests, List Scheduling maintains a list, in which all tasks are ordered according to their priority. Therefore, in order to achieve the scheduling of all tasks, there involves 2 phases in this algorithm: task prioritizing phase and resource allocation phase. In the initial phase, each task will be allocated with a priority according to its property [4]. For example, CPOP(Critical Path on A Processor) employs upward rank and downward rank to determine the priority of each task, while some methods use computation cost. In this way, the tasks with higher priorities will be placed in the front of the list and waiting for execution, while those with lower priorities will be at the back. In the next phase, resource allocation, tasks will be assigned to a proper computing unit for executing, on which the predefined cost can be minimized.

Critical Path on A Processor(CPOP)

As one of List Scheduling algorithms, CPOP has two stages for scheduling.

Task Prioritizing Phase: Firstly, the priority of each task will be set with the sum of upward rank value, $rank_u$, and downward rank value, $rank_d$. And they are defined as below:

$$rank_u(t_i) = \overline{w_i} + \max_{t_j \in succ(t_i)} (\overline{c_{i,j}} + rank_u(t_j))$$

$$rank_d(t_i) = \max_{t_j \in pred(t_i)} (\overline{w_j} + \overline{c_{j,i}} + rank_d(t_j))$$

where $succ(t_i)$ is the set of immediate successor of $task_i$, while $pred(t_i)$ denotes the set of immediate predecessors of $task_i$. Additionally, $\overline{w_i}$ and $\overline{c_{i,j}}$ indicates the computation cost of $task_i$ and communication cost from $task_i$ to $task_j$ respectively. Actually, the sum of downward rank value and upward rank value indicates the length of the path on which the task locates, and all tasks on the critical path have the maximal value. Basically, this also indicates the lower bound for the schedule length, which means the least schedule length.

Resource Allocation Phase: First of all, a dedicated computing unit will be allocated to those tasks locate on the critical path, and the selected computing unit is the one that minimize the cumulative computation time of those critical tasks. Secondly, for the tasks that are not on the critical path, each of them will be allocated to a computing unit on which it can be finished earliest.
Overall, this is a practical method since it is relatively simple. However, it also has some obvious drawbacks that impact on the performance and efficiency. The problem of load unbalancing may arise, as it assigns all tasks on the critical path to a processor. Besides, this would probably also increase the schedule length.

Heterogeneous Earliest Finish Time

The HEFT algorithm is employed to implement scheduling tasks on a heterogeneous environment, in which variant computing resources are deployed. Like other list scheduling algorithms, HEFT also has these two phases, task prioritizing phase and resource allocation phase.

Task Prioritizing Phase: In this phase, the upward rank value is employed to set the priority of each task. Specifically, all tasks will be sorted by decreasing order of $rank_u$. Furthermore, it can be easily seen that the decreasing order of $rank_u$ gives a reveal of the topological order of all tasks. And in this way, it also preserve the precedence constraints, because higher $rank_u$ indicates farther from the entry task.

Resource Allocation Phase: In terms of selection of computing unit, this algorithm is based on an insertion-based policy. More specifically, a task may be inserted to an idle time slot, the gap between execution finish time of a task and execution start time of the subsequent task. And the searching for a proper time slot continues until a capable one was found. Compared with traditional CPOP algorithm, this algorithm outweigh in load balancing, since it allocates tasks to different computing units fairly. Besides, this method also outweigh in schedule length as it utilizes the idle time slots.

2.1.3 Clustering

Clustering is the process in which all tasks of a DAG are mapped onto clusters. To be more specifically, a cluster is a set of tasks that are going to be processed on the same physical computing unit. Initially, there are unlimited number of clusters, thus every task will be assigned to a unique cluster. And then, as the process progresses, some of clusters will be merged and the number of clusters decreases. Finally, the number of clusters will be the same with the number of physical computing units. Therefore, the tasks that are assigned to the same cluster will be executed on the same computing unit. This algorithm involves two phases: cluster merging and and task ordering within a cluster.

2.1.4 Task Duplication

The motivation of proposing this method is to reduce the inter-process communication overhead by simply mapping some of tasks redundantly. Task duplication algorithms are categorized in different types according to their duplication strategy. Although it may reduce inter-process communication significantly, the duplication may cause much duplicated computing tasks and have a negative impact on efficiency.

2.2 Dynamic Scheduling

In previous section, we briefly introduce the static task scheduling algorithms. They are now widely used in parallel and distributed computing. These techniques are utilized in such applications including Gaussian Elimination, Fast Fourier Transformation and Molecular Dynamics Code. Although static scheduling for parallel and distributed computing has been intensively studied for years, it is not a proper method for scheduling in the context of IoT. First of all, in the model of static scheduling, tasks and computing resources are predefined before execution. As mentioned in previous section, tasks are presented in form of a DAG, so scheduling is implemented with the full knowledge of the topology of all tasks. However, it is hard to acquire such information prior to execution. Secondly, in this environment, computing units are shared resources instead of being dedicated ones. Thus, here we conduct a research on dynamic task scheduling algorithms.

In the context of IoT, many terminal devices would be generating computing requests simultaneously and asynchronously. In order to respond such requests on time, and ensure load balancing among devices, scheduling algorithms should be applied to distribute resources properly and efficiently. Above all, there is also a crucial purpose that is to minimize resource starvation, which affects fairness among terminals and the schedule performance. Here we will introduce some conventional scheduling policies in next subsection, after which we will present some improved methods.

2.2.1 Conventional Methods

First of all, **First in First out** is the simplest scheduling policy, under which tasks are simply served in order of arrival. In this method, no queue reorganization is required, so the scheduling overhead is low. On the other hand, since there is no prioritization, it cannot meet the requirement of tasks' deadlines. Secondly, with **Shortest Remaining Time** algorithm, tasks are allocated to resources in which they can be served earliest. This method may be an efficient one in a homogeneous system, but may not be suitable for a heterogeneous system like IoT. The reason is that an early start of execution is not equal to a early finish, especially in a system made of diverse computing units. The processing time may be longer than the waiting time. Lastly, **Fixed Priority Pre-emptive Scheduling** is proposed with the idea that dividing tasks into different priorities, so tasks in a ready queue will be sorted by their priorities. Additionally, lower-priority tasks can be interrupted by incoming higher-priority tasks. This method will be able to ensure the deadlines to be met by assigning these tasks with higher priorities, but it will also bring more risks on starvation when a large number of high-priority tasks are pending in the ready queue.

2.2.2 The heuristics of ECTC and MaxUtil

Apart from above methods, there are also other methods that focus on maximizing resource utilization and reducing energy consumption at the same time. In order to fulfil this goal, ECTC and MaxUtil, two energy-conscious task consolidation heuristics, are proposed by Lee and Zomaya.These two algorithms are based on task consolidation, which is a process of assigning a set of tasks to a set of resources.

With the purpose of reducing energy consumption, this two methods also involve slack reclamation with the support of *Dynamic Voltage/Frequency Scaling*(DVFS) [6]. By the help of this technique, energy-efficiency can be improved by reducing the number of active resources. In this way, the voltage supply level for redundant can temporarily be reduced, so that the cost on energy can be saved.

The key factor if these two factors are their cost functions. Once a task arrives, the heuristic will check every available resource, and then assign this task to the most efficient one. At first, the cost function of ECTC computes the actual energy consumption of the current task subtracting the minimal energy consumption. The cost function is defined below:

$$f_{i,j} = ((p_{\Delta} \times u_j + p_{min}) \times \mu_0) - ((p_{\Delta} \times u_j + p_{min}) \times \mu_1 + p_{\Delta} \times u_j \times \mu_2)$$

Where p_{min} denotes the minimal energy consumption for processing a task, when there are other tasks running in parallel with this one. And p_{Δ} indicates the difference between p_{max} and p_{min}, u_i is the utilization rate of task t_i. As for μ, these three factors including μ_0, μ_1 and μ_2 are the total running time of task t_i, the time in which task t_i is running alone on a computing element and the time when t_i is running in parallel with other tasks, respectively.

Being different from previous method, MaxUtil is proposed to increase the consolidation density, as to reduce energy consumption.The cost function of MaxUtil is defined as:

$$f_{i,j} = \frac{\sum_{\mu=1}^{\mu_0} U_i}{\mu_0}$$

Chapter 3

Designing of System Model and Simulation

3.1 Designing of System Model

In this project, we construct a system model that combines Things devices, Fog devices and Cloud devices. As demonstrated in the figure 3.1, these three categories of devices are deployed from the bottom to the top. Each component of this three-tier system will be described in detail following:

3.1.1 Tier 1: Things

Firstly, on the bottom tier, diverse $Things$ devices are deployed. These devices cover a wide range of wearables, mobile phones, wireless sensor nodes and so on. And they are organized in clusters, each of which is managed by a Fog device. The network connection within a cluster can be treated as LAN connection, thus the network delay is neglected and the bandwidth is enough to carry all connections without contention. Tasks and requests will be generated from these terminals, and each of them is included in an running application. Since $Things$ are embedded in different devices and some of these are even deployed into mobile devices, they only possess limited computing

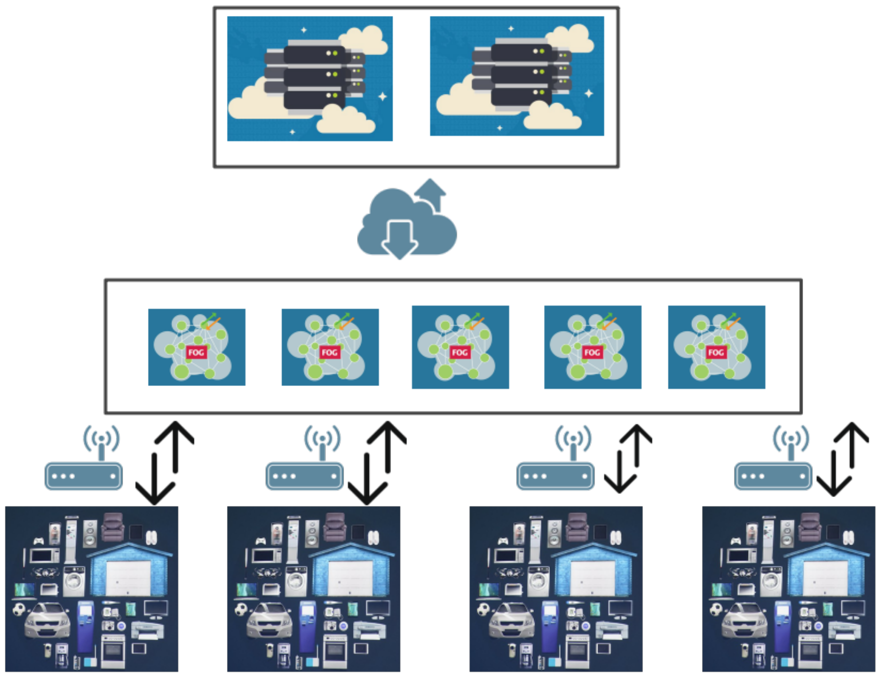

FIGURE 3.1: Three-Tier Architecture of The System Model

resources. Therefore, these generated tasks would be offloaded to upper tier for processing.

3.1.2 Tier 2: Fog

On the second tier, *Fog* devices are deployed. It has been mentioned previously that Fog devices are at the edge of network and act as routers or gateways [7]. Therefore, in this system, Fog devices are designed to be "intelligent" as they not only undertake network management but also be capable of decision making on whether a task should be processed the *Fog* or redirected to *Cloud* [8]. Additionally, with the purpose of ensure load balancing, once local *Fog* was busy, request can also be redirected to a neighbour *Fog* server which is geographically closer. *Fog* servers are connected via Internet. Above all, as for the interior architecture, we assume that every fog server is equipped with a SIMD optimized processor and the speedup ratio is 3.

3.1.3 Tier 3: Cloud

Lastly, on the top tier, there deployed Cloud servers. Cloud servers are defined with the characteristics of high performance and the ability of processing diverse tasks, and they are accessed via Internet. In the environment of this simulation, there are two cloud servers in this system, and these two cloud server share the same computing capability. Above all, cloud servers can process parallel application by using multiple computing elements, and a single cloud server can utilize at most 8 processors to serve one task concurrently.

As described above, each of these three tiers has its unique properties. Therefore, we utilize different components for different tasks. Firstly, as Fog devices are deployed at the edge of network and can provide a short latency, they are mainly used to process light tasks and real-time tasks. On the other hand, they can provide neither powerful or heterogeneous computing ability. In this case, those heavy tasks which require intensive computing would be offloaded to Cloud servers. And in this way, the powerful and heterogeneous computing ability of Cloud devices can be fully utilized.

In this system, different components are fully connected, thus there is no network contention. With the purpose of simulating the real condition, we build this model of 5 Fog devices and 2 Cloud devices.

3.2 Designing of Application Model

3.2.1 Characteristics of Tasks

Based on previous research [9], in our experiment, tasks arrive in a Poisson process, and the computation cost follows exponential distribution.

Task Type	Accelerate By	Percentage	Example of Application
Serial Tasks	None	60%	Simple computation tasks and controlling instructions.
SIMD Tasks	GPGPU	25%	Image processing and Scientific computation.
Parallel Tasks	Cluster	15%	Complex computation and data analysis.

TABLE 3.1: Classification of Tasks

3.2.2 Classification of Tasks

As the motivation of introducing this computing paradigm is to undertaking different types of computing tasks generated from $Things$ in an universal platform, here we classify the computing tasks in following categories:

- **General Purpose Tasks:**
  This type of tasks can also be treated as serial computation tasks. In the traditional way of programming, in order to solve a problem, an algorithm is constructed as a serial stream of instructions. In this way, these tasks cannot facilitate parallel computing, to accelerate processing speed. This type of tasks account a significant proportion of IoT tasks, including simple computation requests and control commands.

- **Single Instruction Multiple Data-flow Tasks:**
  SIMD is used to describe the computers that contains multiple computing units to perform the same operation in parallel. And also, as its name suggests, a SIMD instruction will generate multiple operands or data-flow for computation. Therefore, it can facilitate a single SIMD-optimized processor like GPGPU or optimized CPU to accelerate processing. This type of task covers a wide range including multimedia application and scientific computation.

- **Parallel Computation Tasks:** Unlike conventional serial computation tasks, the principle of parallel computing is a problem is divided into a number of smaller ones. In this manner, this type of tasks can facilitate parallel and distributed computing system which contains many individual computing elements.

3.3 Designing of Task Scheduling Policy

As to provide with better QoS, inspired by A* algorithm [10] in shortest path routing problem, We propose such dynamic task scheduling policy with heuristic as below.

$$F_{i,j} = D_j + H_{i,j}$$

Where $F_{i,j}$ is the cost function of task t_i running on computing resource r_j. And then, D_j is the earliest time when device r_j will be available for another task. After this, the heuristic $H_{i,j}$ signifies the estimated processing time consumed to finish task t_i by device r_j. Overall, this cost function indicates the earliest finish time for task t_i when processed on r_j. Before being assigning to a computing resource, the system will go through this calculation to determine which resource will finish this task in the earliest time.

3.4 Quality of Service

In order to measure the performance and QoS, here we introduce some metrics for evaluation among these different approaches.

- **makespan of all applications**
  This metric can also be considered as schedule length, which indicates the duration from the start of the first task to the end of the last task. Alternatively, it can also be defined as the maximal value of execution finish time among all tasks.

  $$makespan = \max_{t_i \in T}\{FT(t_i)\}$$

  Where FT denotes the finish time of task t_i, and T is the set of all tasks. This metric presents the actual duration spent on processing tasks. And also, it is the response time for finishing all applications.

- **Cumulated Running Time**

  $$CTime = \sum_{t=0}^{N}\{CompT(t_i) + CommT(t_i)\}$$

FIGURE 3.2: Simpy: The Framework for Simulation

Where $CompT$ denotes the time spent on processing, and $CommT$ signifies the time spent on communication. This metric indicates the cumulated running time of all tasks. This metric is consist of processing times and communication times of all tasks.

3.5 Environment

The simulation of this project is implemented on Simpy, a process-based discrete event simulation framework based on Python.

Chapter 4

Experimental Evaluation

4.1 Evaluation on The Combination of Fog and Cloud

In this section, we initially evaluate the performance improvements brought by the combination of Fog and Cloud compared with the scenario with only Fog and only Cloud.

4.1.1 Search for Optimal Threshold

Here we define the term $Threshold$, it means the point of computation cost beyond which the task will be offloaded to cloud. In following figure 4.1, the vertical axis indicates the cumulated running time of all tasks, and the horizontal axis indicates the threshold. In other words, on the horizontal axis, more tasks will be sent to fog in the positive direction.

As we can see from the figure, the cumulated time decreases as the threshold increases, so it can be concluded that the involvement of fog brings slight performance improvement. And the it reaches the bottom at the point of 25000, where the system costs the minimal time on processing all tasks. On the other hand, as the threshold increases beyond this point, the cumulated running time rises sharply.

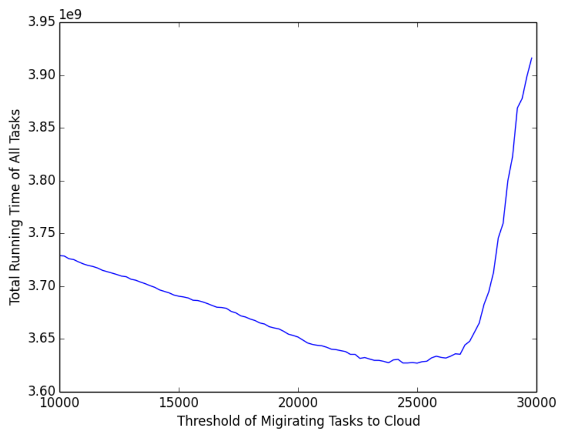

FIGURE 4.1: Experiments on Different Offloading Threshold

This indicates that excessive amount of task being sent to fog brings the huge increase of cumulated running time. As for the cause of this super-linear growth, it probably be caused by excessive incoming tasks most of which are blocked in the queue. As fog devices are simply routers and gateways, they are not able to handle so many tasks. Therefore, when excessive tasks arrive, a large proportion of them will be blocked in the pending queue or even discarded.

4.1.2 Improvements on Cumulated Running Time

In this section, we conduct an experiment to evaluate the improvements on cumulated running time when the combination of fog and cloud applies, and with the offload threshold obtained in previous section. As illustrated in figure4.2, when all tasks are sent to fog, the cumulated running time goes up in a super-linear manner, because the arriving rate exceeds the processing rate of fog device and then a large number of them would be blocked in the queue. At the same time, it is also demonstrated that the time increases slower when all task were sent to cloud. Lastly, when we apply the offload threshold to this system, the growth is minimal among all conditions.

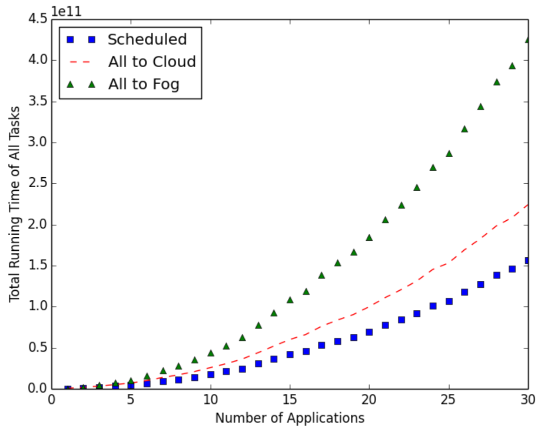

FIGURE 4.2: Cumulated Running Time on Different Policies

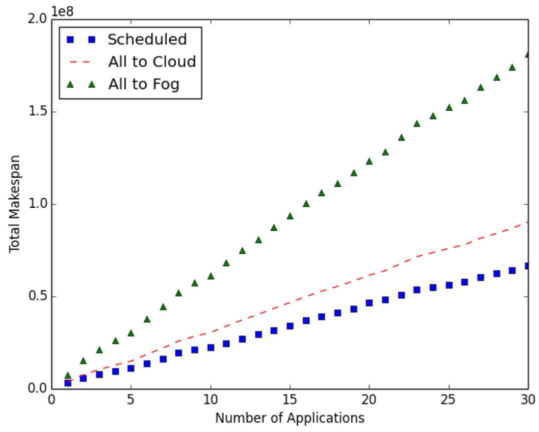

FIGURE 4.3: Makespan on Different Policies

4.1.3 Improvements on Makespan

It is indicated in figure 4.3 that makespans in these three conditions all increase linearly. On the other hand, when fog and cloud both involve in this system, the growth rate is minimal among all. It can be concluded that the involvement of the combination make the system finish all tasks in a shorter time.

4.2 Evaluation on Heterogeneous System

At this stage, we conduct an experiment on the heterogeneous system. As can be seen from the figure 4.4, the cumulated running time

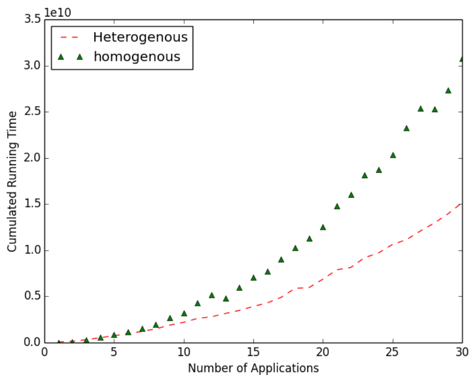

FIGURE 4.4: Cumulated Running Time on Homogeneous and Heterogeneous

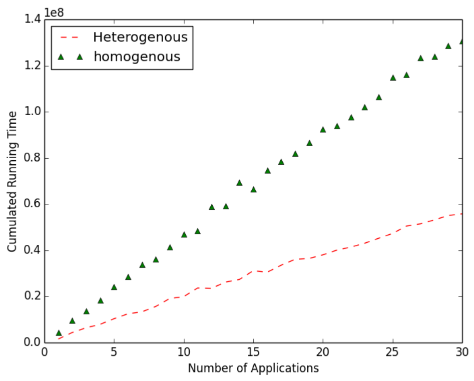

FIGURE 4.5: Makespan on Homogeneous and Heterogeneous

increases slower on heterogeneous system. Furthermore, as the number of application rises, the gap between homogeneous and heterogeneous continues to widen. In other words, the speedup increase as the number of application increases. Above all, the reduced cumulated running time also means that the less tasks will be blocked in the pending queue. Besides, with less task waiting in cache of devices, less data will be resent due to package loss, thus it may also help reduce network traffic.

In terms of makespan, our proposed approach also shows a great advantage over the original. When there are 30 applications running in parallel, the speedup is approximately 3. This means that the heterogeneous system with scheduling policy cost only one-third of time to finish all tasks compared to the traditional one.

Chapter 5

Conclusion

5.1 Conclusion on Experiments

In this research, having conducted such experiments on the platform which combines Things Fog and Cloud, we drawn following conclusions.

Firstly, by schedule tasks according to their types, computation cost and estimated finish time on each resource, and also with the help of offloading policy, both response time and cumulated running time are improved. In this way, applications can be provided with better QoS, and end users can have better experience. Especially, for those real-time applications, which have strict requirements on response time, it is clear that fog computing is pivotal to meet their requirements.

At the second stage of experiment, on the heterogeneous platform we built, the experiment result demonstrates a significant improvement brought by heterogeneous architecture and the scheduling policy. Moreover, the speedup increases with the increase of amount of applications, so this architecture and the scheduling policy will show a greater superiority over original system. This indicates that this approach can enable more applications to run concurrently.At the same time, it also means that less energy would be consumed for processing the same amount of tasks, this is also a progress in green computing and sustainable development. Above all, this improvement does not cost extra investment as GPGPU deployed on network edge is not expensive and cloud is naturally built for parallel applications.

5.2 Future Works

In this research, we conduct experiments on a simple system, in which network congestion and precedence between tasks are not considered. Considering these factors, in the future, we are planning to employ a more realistic model to investigate the performance of different Task scheduling approaches. Firstly, precedence constraints between task are to be involved by organizing them in the form of DAG, which is supposed to reflect the real situation. Secondly, we are going to introduce network model that is closer to the real environment, in which the impact of network congestion can be illustrated. Lastly, apart from cumulated running time and makespan, more QoS factors will be utilized to measure the performance, which includes Queue length and energy consumption.

Bibliography

[1] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of internet of things",

[2] S. Baskiyar and C. Dickinson, "Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors using task duplication", *Journal of Parallel and Distributed Computing*, vol. 65, no. 8, pp. 911–921, 2005.

[3] G. Xie, R. Li, X. Xiao, and Y. Chen, "A high-performance dag task scheduling algorithm for heterogeneous networked embedded systems", in *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, IEEE, 2014, pp. 1011–1016.

[4] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing", *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 260–274, 2002.

[5] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems", *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012.

[6] T. Kuroda, K. Suzuki, S. Mita, T. Fujita, F. Yamane, F. Sano, A. Chiba, Y. Watanabe, K. Matsuda, T. Maeda, *et al.*, "Variable supply-voltage scheme for low-power high-speed cmos digital design", *Solid-State Circuits, IEEE Journal of*, vol. 33, no. 3, pp. 454–462, 1998.

[7] R. Deng, R. Lu, C. Lai, and T. H. Luan, "Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing", in *Communications (ICC), 2015 IEEE International Conference on*, IEEE, 2015, pp. 3909–3914.

[8] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things”, in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, ACM, 2012, pp. 13–16.

[9] H. Khazaei, J. Mišić, and V. B. Mišić, “Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems”, *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 5, pp. 936–943, 2012.

[10] S. Russell, P. Norvig, and A. Intelligence, “A modern approach”, *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, p. 27, 1995.

Appendix A

Simulation Program Code

```
import simpy
import random
import matplotlib.pyplot as plt
import numpy

Types = ['general', 'simd', 'para']#task types

env = simpy.Environment()#running environmrnt
#repeat simulation for times at different ratio
envs = []
#an array to store all experiment results
res = []
horizontal = []

Threshold = 20000
App_num=[i for i in range(1,31)]
#lifeTime_fog = 0
#lifeTime_cloud = 0
abc = []
#makespan = [0]*App_num
makespan = [0]*30
fog_status = 5*[0]
remaining_time = 5*[0]
qlen = 5*[0]
```

```
#utilization_cloud = [None]*2

def TaskGenerator(env, num, pipe, app_num, start):

    """Tasks generated from here"""

    for i in range(1,num):
        yield env.timeout(100)
        name = 'Task␣%d'%(i)
        #dataSize = DSizes[start+i-1]
        #computationCost = CompSizes[start+i-1]
        rnum = random.randint(1,100)
        if rnum>=40:
            TaskType = 'general'
        elif rnum>=15:
            TaskType = 'simd'
        else:
            TaskType = 'para'

        #TaskType = random.choice(Types)
        TimeStamp = env.now
        appNum = app_num

        if TaskType=='simd':
            computationCost = 5*CompSizes[start+i-1]
            dataSize = 3*DSizes[start+i-1]
        elif TaskType=='para':
            computationCost = 10*CompSizes[start+i-1]
            dataSize = 8*DSizes[start+i-1]
        else:
            computationCost = CompSizes[start+i-1]
            dataSize = DSizes[start+i-1]

        if i==num-1:
            flag = True
        else:
            flag=False
        task = {'Name':name, 'ComputationCost':computationCost, '
'LastOne':flag}

        yield pipe.put(task)

def data_transfer(env, pipe, pipe_fog, pipe_cloud, threshold):

    while True:
        BandWidth = 20000
        task = yield pipe.get()
        computationCost = task['ComputationCost']
        dataSize = task['DataSize']

        if computationCost>threshold:#offload to cloud
            task['TimeStamp'] = task['TimeStamp'] - dataSize*50
            yield env.timeout(dataSize/BandWidth)
            yield pipe_cloud.put(task)
        else:
            task['TimeStamp'] = task['TimeStamp'] - dataSize*4
            yield env.timeout(dataSize/BandWidth)
            #print env.now
            n = random.randint(1,100)%5
            yield pipe_fog[n].put(task)

        #yield env.timeout(dataSize/BandWidth)

def data_transferIMPV(env, pipe, pipe_fog, pipe_cloud, threshold)
    while True:
        BandWidth =20000
        task =yield pipe.get()
        computationCost = task['ComputationCost']
        dataSize = task['DataSize']

        if task['TaskType']=='para':
            task['TimeStamp'] =task['TimeStamp'] - dataSize*50
            yield env.timeout(dataSize/BandWidth)
            yield pipe_cloud.put(task)
        elif task['TaskType']=='simd':
            if 0 in fog_status:
                fog_id = fog_status.index(0)
                task['TimeStamp'] = task['TimeStamp'] - dataSize*
                yield env.timeout(dataSize/BandWidth)
                yield pipe_fog[fog_id].put(task)
            else:
                task['TimeStamp'] = task['TimeStamp'] - dataSize*
                yield env.timeout(dataSize/BandWidth)
                yield pipe_cloud.put(task)
        else:
            task['TimeStamp'] = task['TimeStamp'] - dataSize*4
            yield env.timeout(dataSize/BandWidth)
            fog_id = random.randint(0,4)
            yield pipe_fog[fog_id].put(task)

def cloud(env, pipe_cloud):

    global lifeTime_cloud
    global makespan

    while True:
        task = yield pipe_cloud.get()
        if task['TaskType']=='para':
            compTime = task['ComputationCost']*1/8
            yield env.timeout(compTime)
        elif task['TaskType']=='simd':
            compTime = task['ComputationCost']*1/3
            yield env.timeout(compTime)
        else:
            compTime = task['ComputationCost']*1
            yield env.timeout(compTime)

        lifeTime_cloud = lifeTime_cloud + (env.now - task['TimeSta
        #if name=='Name %d'%(num-1):
        #     makespan = env.now
        if task['LastOne']==True:
            seq = task['APPNUM']-1
            makespan[seq]=env.now
            #print 'task %d finished at %d'%(task['APPNUM'], env.

def cloudIMPV(env, pipe_cloud):
    global lifeTime_cloud
    global makespan

    while True:
        task = yield pipe_cloud.get()
        if task['TaskType']=='para':
            compTime = task['ComputationCost']*1/3
            yield env.timeout(compTime)
            lifeTime_cloud = lifeTime_cloud + (env.now - task['Tir
        else:
            compTime = task['ComputationCost']*1/8
            yield env.timeout(compTime)
            lifeTime_cloud = lifeTime_cloud + (env.now - task['Tir
        if task['LastOne']==True:
            seq = task['APPNUM']-1
            makespan[seq]=env.now

def fogIMPV(env, pipe_fog, device_ID):

    global lifeTime_fog
    global makespan

    while True:
        task = yield pipe_fog[device_ID].get()
        fog_status[device_ID]=1
        if task['TaskType']=='simd':
            compTime = task['ComputationCost']*5/3
            gap = env.now - task['TimeStamp']
            yield env.timeout(compTime)
            lifeTime_fog = lifeTime_fog+(env.now - task['TimeStam
        else:
            compTime= task['ComputationCost']*5
            gap = env.now - task['TimeStamp']
            yield env.timeout(compTime)
            lifeTime_fog = lifeTime_fog+(env.now-task['TimeStamp'
        fog_status[device_ID]=0


def fog(env, pipe_fog):
    global lifeTime_fog
    global makespan

    while True:
        task = yield pipe_fog.get()
        if task['TaskType']=='simd':
            compTime = task['ComputationCost']*3/3
            gap = env.now -task['TimeStamp']
            yield env.timeout(compTime)
        else:
            compTime =task['ComputationCost']*3
            gap = env.now - task['TimeStamp']
            yield env.timeout(compTime)

        lifeTime_fog = lifeTime_fog+(env.now - task['TimeStamp'])

        if task['LastOne']==True:
            makespan[task['APPNUM']-1]=env.now

for i in range(1,31):#the experiment repeats 30 times
    envs.append(simpy.Environment())

#index = 0
taskNum = [0]*30
CompSizes = []
DSizes = []
for i in range(0,30):#10 apps run in paralell
    taskNum[i] = random.randint(100,200)

for i in range(1,31):
    for x in numpy.random.exponential(scale=20000, size=taskNum[i
        CompSizes.append(x)
        DSizes.append(x*random.randint(20,30)/10)
```

```
#set the start point of each group of numbers
points = []
points.append(0)

for i in range(1,30):
    points.append(sum(taskNum[0:i]))

index = 0

makespan = [0]*30
horizontal1=[]
res1=[]
#plt.plot(horizontal,res, 'bs', label='Scheduled')

envs1 = []#heterogenous system
for i in range(1,31):
    envs1.append(simpy.Environment())
index = 0
for e in envs1:
    #makespan = 0
    lifeTime_fog = 0
    lifeTime_cloud =0
    pipe = simpy.Store(e)
    #pipe_fog = [simpy.Store(e)]*5
    #pipe_fog = simpy.Store(e)
    pipe_fog = 5*[simpy.Store(e)]
    pipe_cloud = simpy.Store(e)

    for i in range(App_num[index]):#generate 10 apps
        e.process(TaskGenerator(e, taskNum[i], pipe, i, points[i]
    #e.process(TaskGenerator(e, taskAmount, pipe,0))
    e.process(data_transferIMPV(e, pipe, pipe_fog, pipe_cloud,0))
    f_0 = e.process(fogIMPV(e, pipe_fog, 0))
    f_1 = e.process(fogIMPV(e, pipe_fog, 1))
    f_2 = e.process(fogIMPV(e, pipe_fog, 2))
    f_3 = e.process(fogIMPV(e, pipe_fog, 3))
    f_4 = e.process(fogIMPV(e, pipe_fog, 4))
    c_0 = e.process(cloudIMPV(e, pipe_cloud))
    #c_1 = e.process(cloud(e,pipe_cloud))
    e.run(until=None)
    print 'Total␣makespan␣is␣%.2f␣for␣processing␣%d␣applicatinos'
    #print 'Threshold is %d'%(Threshold)
    res1.append((lifeTime_fog+lifeTime_cloud)/10)
    #res1.append(e.now)
    horizontal1.append(index+1)
    index = index +1

print 'The␣standard␣deviation␣is␣%.2f'%(numpy.mean(makespan))

plt.plot(horizontal1,res1,'r--', label='Heterogenous')



makespan = [0]*30
envs2=[]
for i in range(1,31):
    envs2.append(simpy.Environment())

horizontal2 = []
res2 = []
index = 0
for e in envs2:
    #makespan = 0
    lifeTime_fog = 0
    lifeTime_cloud =0
    pipe = simpy.Store(e)
    #pipe_fog = [simpy.Store(e)]*5
    #pipe_fog = simpy.Store(e)
    pipe_fog = 5*[simpy.Store(e)]
    pipe_cloud = simpy.Store(e)

    for i in range(App_num[index]):#generate 10 apps
        e.process(TaskGenerator(e, taskNum[i], pipe, i, points[i]
    #e.process(TaskGenerator(e, taskAmount, pipe,0))
    e.process(data_transfer(e, pipe, pipe_fog, pipe_cloud,1000000
    f_0 = e.process(fog(e, pipe_fog[0]))
    f_1 = e.process(fog(e, pipe_fog[1]))
    f_2 = e.process(fog(e, pipe_fog[2]))
    f_3 = e.process(fog(e, pipe_fog[3]))
    f_4 = e.process(fog(e, pipe_fog[4]))
    c_0 = e.process(cloud(e, pipe_cloud))
    #c_1 = e.process(cloud(e,pipe_cloud))
    e.run(until=None)
    print 'Total␣makespan␣is␣%.2f␣for␣processing␣%d␣applicatinos'
    #print 'Threshold is %d'%(Threshold)
    res2.append((lifeTime_fog+lifeTime_cloud)/10)
    #res2.append((e.now))
    horizontal2.append(index+1)
    index = index +1


plt.plot(horizontal2,res2,'g^', label='homogenous')
plt.legend(loc='upper␣left')
plt.xlabel('Number␣of␣Applications')
plt.ylabel('Cumulated␣Running␣Time')
plt.show()
```