UNIVERSITÀ DELLA CALABRIA

Dipartimento di Ingegneria Informatics, Modellistica, Elettronica E Sistemistica

Dottorato di Ricerca in
Ricerca Operativa
XXIV ciclo

Tesi di Dottorato

Optimized Dynamic Load Balancing in Distributed Exascale Computing Systems

Seyedeh Leili Mirtaheri

D.I.M.E.S., Novembre 2015
Settore Scientifico Disciplinare: MATH/09

UNIVERSITÀ DELLA CALABRIA

Dipartimento di Elettronica,
Informatica e Sistemistica

Dottorato di Ricerca in
Ricerca Operativa
XXIV ciclo

Tesi di Dottorato

Optimized Dynamic Load Balancing in Distributed Exascale Computing Systems

Seyedeh Leili Mirtaheri

Supervisor:

Professor Lucio Grandinetti

DIMES- DIPARTIMENTO DI INGEGNERIA INFORMATICS, MODELLISTICA, ELETTRONICA E SISTEMISTICA
November

Settore Scientifico Disciplinare: MAT/09

Abstract

The dynamic nature of new generation scientific problems needs undergoing review in the traditional and static management of computing resources in Exascale computing systems. Doing so will support dynamic and unpredictable requests of the scientific programs for different type of resources. To achieve this facility, it is necessary to present a dynamic load balancing model to manage the load of the system efficiently based on the requests of the programs. Currently, the distributed Exascale systems with heterogeneous resources are the best branch of distributed computing systems that should be able to support the scientific programs with dynamic and interactive requests to resources. In this thesis, distributed Exascale systems are regarded as the operational and real distributed systems, and the dynamic load balancing model for the distributed controlling of load in the nodes in distributed Exascale computing systems are presented. The dominant paradigm in this model is derived from Operation Research sciences, and the request aware approach is replaced with the command-based approach in managing the load of the system. The results of evaluation show us the significant improvement regarding the performance by using the proposed load balancing mechanism in compare with the common distributed load balancing mechanisms.

Rende, Cosenza, Italy *Seyedeh Leili Mirtaheri*

November 2015

Contents

1 Introduction

Distributed system is a fascinating field that is evolving rapidly across many domains. Classical systems and learning theories have focused on optimizing stand-alone systems or learners with great success [1].

These studies have brought forward notable examples of complex systems that derive their sophistication from coordination among simpler units and from the aggregation and processing of distributed pieces of information. While each unit in these systems is not capable of sophisticated behavior on its own, it is the interaction among the constituents that leads to systems that are resilient to failure and that are capable of adjusting their behavior in response to changes in their environment. These discoveries have motivated diligent efforts towards a deeper understanding of processing, adaptation, and learning over complex networks in several disciplines including machine learning, optimization, control, economics, biological sciences, information sciences, and the social sciences [2, 3].

In the distributed mode of operation, nodes are connected by a topology and they are permitted to share information of resources only with their immediate neighbors [4]. There are many good reasons for the peaked interest in such distributed solutions. Some of these reasons have to do with the benefits of cooperation in terms of improved performance and improved robustness and resilience to failure. Other reasons deal with privacy and secrecy considerations where nodes may not be comfortable sharing their data with remote fusion centers. In other situations, the resources may already be available in dispersed locations, as happens with distributed computing systems such as grid computing, cloud computing and distributed Exascale computing [5, 6].

Distributed computing system consists of a group of computers, each of which has an independent operating system and all are connected to each other through a computer network [7]. There is a system software in each of these computers that envisages the whole distributed system as a coherent system to its users. Each user is assumed to be able to connect to and transparently access all available resources in the distributed system as though connecting and accessing their own local resources. This definition of distributed computing systems is very general and covers all types of purposes for user resource connectivity. Users get connected to resources to receive different facilities and services like computing, file sharing, I/O sharing, and storage. In distributed computing systems with mission of high performance computing (HPC), the main concern of managing resources is reaching to minimum execution time in running scientific/industrial programs [8, 9].

Distributed Exascale computing system (DECS) in this thesis is defines as a distributed computing system with concerns of Exascale computing. In DECSs, efficient execution of program is directly related to effective management of resources. Many attempts have been done to improve effective utilization of resources and efficient management mechanisms have been proposed to solve the challenges during the time. In resource management framework, the load balancer is the key unit in efficient utilization of resources and improving response time.

In the 21st century, the HPC community has experienced astronomical increases in the complexity and sensitivity of scientific and commercial problems to be solved by the computer application programs. Exascale computing is a promising technology for satisfying such demands and their challenges [5, 9].

In roadmap of Exascale computing as the next generation HPC systems, is mentioned that the scientific programs are going to get more complex with unpredictable requests and requires large scale and more powerful HPC systems [5].

These challenges should be concerned in design and development of next generation high performance computing systems. It means all unites of next generation HPC systems should be designed and developed based on the behavior of the new scientific programs. As mentioned, load balancer unit has pivotal role is HPC systems and as the main software units of these systems, should be designed and developed respect to mentioned challenges [8, 10].

We consider important challenges of load balancing in DECSs and we focus on its two major aspects as follows: 1) how to deal with complex nature of 21 century problems and unpredictable behavior of them 2) how to deal with scalable, dynamic and distributed platforms [11].

The challenge of complex nature of new scientific problems means the program has dynamic and unpredictable requirements at runtime, it cause the load of the system changes dynamically.

The challenge of scalable, dynamic and distributed platform means there is not a stable and predictable underlying platform and there is not any guaranty for efficient load execution after assigning the loads to nodes so we have to balance the load of the system at runtime dynamically.

We consider the issue of developing a mechanism for estimating the load status in nodes in a distributed computing system over a partially connected communication network and balance it.

In load balancing mechanism, there are some parameters that they have direct impact in efficiency of load balancing operation. A group of these parameters are determined by specifying the status of nodes [12]. Practically, the nodes’ speed up in executing the commands and the current load of them are two important factors in specifying the status of the nodes and making decision to distribute the load based on them. Actually, it’s clear that, the load balancer should migrate and transfer the extra load from the nodes with low speed and over loaded to nodes with high speed and lower loaded. Therefore, defining the parameters that consider the both the factors of commands execution speed up and the load of the nodes and using them in calculating the status of the nodes can be effective in making accurate decision in load balancing unit.

The second group of parameters are related to execution of load balancing mechanism that among them specifying the start time of load balancing mechanism is important issue. Different strategies are presented for this issue but in this thesis we use one-shot strategy. In this policy, at the moment that extra load is imposed in the node, the load balancing operation start to work [13].

The third group of parameters are related to interconnection platform status. The important effective parameter in load balancing mechanism from these parameters is the communication delay between the nodes. In this thesis, we focus on this parameter in proposing the efficient load balancing mechanism. Actually, in load balancing mechanism, two types of information transfer are done. The first types are the information of node status, command execution speed up and the current load of the nodes. Since the amount and volume of these information is low, therefore, the system does not impose significant delay derived of these type of information transfer. But the second type of information is related to the load that should be transferred. Since the volume of transferred load in most of the programs is too much, the delay caused by the transmission will be most significant and it’s directly negative impact on system performance. In this thesis, a Compensating factor is defined to transfer accurate portion of load to neighbor nodes by considering the transfer delay. We also suggest three methods to determine the value of this factor to transfer accurate amount of load [14].

The suggested load balancing mechanism in this thesis is completely distributed because it is designed to work in Exascale computing systems. One of the important issue in this area is the system's scalability and in centralized architecture, it is completely limited by the server node capacity [15].

Actually, in the suggested load balancing mechanism, suggested in this thesis, there are not any centralized operations. All the operations such as gathering the information of node status, distributing the load, result gathering are done completely locally and distributed. It means that all the nodes make decision on all the operations themselves, cooperating with their neighbors and finally, the total load of the system is balanced. Distributed computing systems offer an efficient and inexpensive way to process workloads composed of a large number of independent tasks in a cooperative manner.

Based on status of nodes connections, distributed systems are divided into two groups named fully connected and partially connected distributed systems [16]. In fully connected distributed systems, each node has access to all other nodes in the system directly while in partially connected distributed systems there is a concept named neighborhood and each node only has direct access only to its neighbors. Actually, the real implemented distributed systems are partially connected network and the considered distributed system in this thesis is partially connected systems with support of scalability in size, geography and management.

In distributed system, there are two strategies for distributing the loads among the nodes, named sender-initiated strategy and receiver initiated strategy [17]. In sender based strategy, the processor is overloaded, and sends work to another that initiates the transaction. In receiver base, the under-loaded processor initiates the transaction by sending a message to other nodes. In some cases, receiver based algorithms are better and in some cases, depending on the algorithm and network topology, sender based may sometimes be better. In this thesis we used sender initiated strategy based on distributed platform.

In this thesis, a model is proposed that each node can calculate the number of available tasks in their inside at the specific time period and each node can estimate its status based on this model. Each node by calculating this model and estimating its status will be member of two following sets that they are defined in this thesis, I and I^c. I is a set with member of nodes with positive load, and I^c is a set of nodes with negative load. The nodes with negative load, relative to their computing power and volume of load that is intended to them to run, have more capacity to execute the tasks and it's possible to assign more task to them. On the other hand, the nodes with positive load has less computing power than the considered load for them and a portion of their load should be transferred to other nodes. Practically, this load balancing mechanism can be introduced for estimating amount of transferred load from node with positive load to node with negative load.

The next phase, core of the thesis, has aimed at showing the strong relation between load balancing mechanism and operations research. In particular, it has explained how to adapt and extend the classic models and methodologies provided by operations research for efficiently solving specific challenging issues related to load balancing. In this direction, we have focused on the question that how specify the optimum amount of transferred load while the idle times of nodes in the system be in its minimum states? We present a model for accurate estimating of extra load in each node. There are two important issues in this model, the first issue is that model should be used distributed and second issue is that the delay time of load transferring should be considered in this model. In the proposed model, the delay of load transferring is considered as an important issue in making decision about the migration of load from one node to another node. The transfer delay factor is considered in the proposed model in specifying the portion of transferring load while the performance of the proposed load balancing mechanism increase, especially in the systems that the delay of transferring is noticeable. In the evaluation phase, the following issue are evaluated:

– The task execution process versus the time in the network when load balancing is not performed

– The task execution process versus the time in the network when load balancing is performed without load transfer delay considerations

– The effect of load transfer delay on idle time of systems.

– The performance evaluation of our first proposed method on total task execution time

– The illustration of load execution time where source and destination nodes have different processing speed and no compensation is used.

– Performance evaluation of first proposed method on total load execution time where source and destination nodes have different processing speed and no compensation is used.

– Performance evaluation of second proposed method on total load execution time where source and destination nodes have different processing speed and no compensation is used.

– Effect of compensator value in the total execution time

– Effect of different load transfer speed on the load execution time.

– Effect of different load execution speed on the load execution time

– The effect of execution speed up ration and load transferring speed up in execution of the load

– The impact of network size in performance of proposed methods.

The results show us the significant improvement in performance by using the proposed load balancing mechanism in compare with the common distributed load balancing mechanisms.

The thesis is organized as follows. Chapter 2 contains a background overview of load Balancing, workload definition, load balancing metrics, dynamic load balancing, the policies of dynamic load balancing algorithms, centralized and distributed based dynamic load balancing, centralized dynamic load balancing, primary and centralized node based dynamic load balancing, distributed non-cooperative dynamic load balancing, distributed cooperative dynamic load balancing, load balancing in distributed Exascale computing systems. In Chapter 3 we propose a novel dynamic and distributed load balancing model. The model present a model to estimate the machine state locally and then specify how much of load should be transferred. In Chapter 4 simulation results of the average completion time of a workload using the proposed model along with the extended global one-shot DLB policy are presented. We also compare the proposed model with other load balancing mechanism and present the results of evaluation. Our conclusions and suggestions for future work are presented in Chapter 5.

2 Background

This chapter presents some background for the research presented in this thesis. We start with a review, in Sections 2.1, of load balancing concept and general issues, workload definition and load balancing metrics, which provides an important context for load balancing area. In Section 2.2 we discuss load balancing categories and give a good category of load balancing with approach of practically. In Section 2.3 we review state-of-the-art algorithms for dynamic load balancing.

2.1 Load Balancing

Load balancing is a method for distributing tasks onto multiple computers based on different policies [18]. For instance, distributing incoming HTTP requests (tasks) for a web application onto multiple web servers. In parallel computing systems, when a parallel job is assigned to the system for execution, load balancer specifies the time and location of each parts of the parallel job to execute based on the global objective and introduced executive policies in the system. In high performance computing systems (HPCS), the main objective are decreasing response time and increasing throughput [19]. Figure 2.1 is a diagram illustrating the basic principle of load balancing:

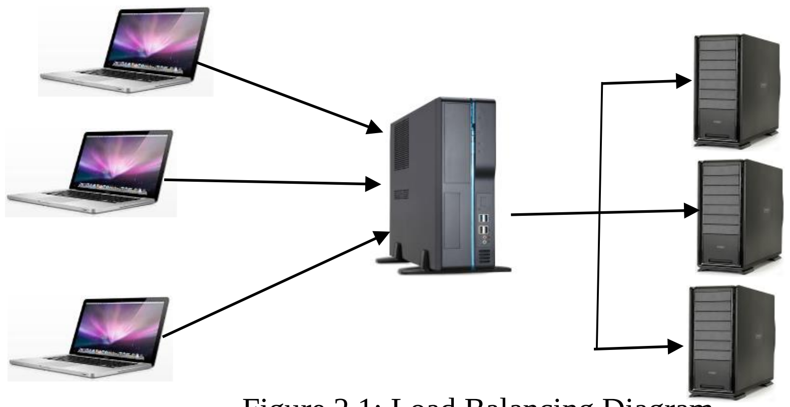


Figure 2.1: Load Balancing Diagram

Standard load balancing manager includes five activities, queuing, scheduling, monitoring, resource control and accountants. As shown in figure 2.2, Queuing is the first phase of load balancing operation and in this unit the tasks assigned by user, and this process finished by giving the result of executed program to user [20].

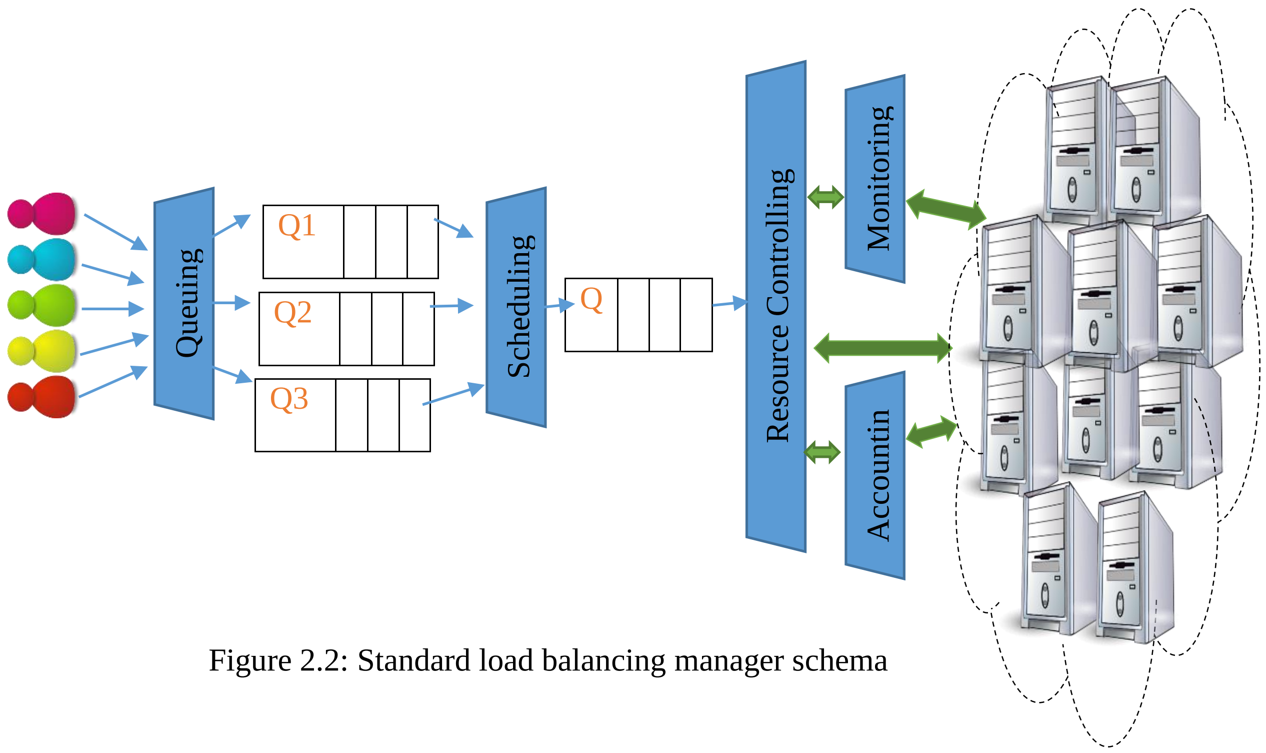


Figure 2.2: Standard load balancing manager schema

The aim of these five activities is efficient matching of user-specified workload to existing resources by keeping balance the load of the system. As shown in figure 2.2, the standard process of load balancing manager starts by receiving the jobs from users in queuing unit. Different people from separate locations at various times submit different jobs to the system that are possibly ignorant of needs being executed on the same system by other users. A queuing unit put the randomly submitted jobs to queues until system resources are available to assign to each of them. Different distributed queues may be provided based on system aim and policies, priorities and specific requirements to reach optimal scheduling [20, 21].

After placement of jobs in queues, scheduling unit is activated. Scheduling is a fundamental technique for improving performance in computer systems. Scheduling policies are implicitly (or explicitly) used everywhere that a resource needs to be allocated. Whenever there is contention for a limited resource, a queue builds, and a scheduling policy is used to determine the order in which the resource is allocated to satisfy requests. The scheduler has to balance the priorities of each job, with the demands of other jobs, the existing system compute and storage resources, and the governing policies dictated for their use by system administrators. Scheduler should be able to deal with different types of jobs such as huge jobs, small jobs, real-time jobs, high-priority jobs, dynamic jobs, static jobs and etc. The scheduler governs the order of execution based on these independent priority assessments [22].

In this architecture, existence of a unit for controlling the resources and putting the programs on the designated nodes, moving the necessary files to the respective nodes, starting jobs, suspending jobs, terminating jobs, and offloading result files is necessary. Resource controlling unit is responsible of mentioned duties. It informs the scheduler when resources are available and handles any exception conditions across the set of nodes committed to a given user job [20].

Since gathering information about status of resources help to make decision accurately, a unit should be designed in this architecture to track resource availability, task status on each node, and operational health of the nodes continuously. Monitoring can be active based on polling method or interrupt method [23].

Other unit of the architecture is accounting unit. Actually, in distributed computing systems, records must be automatically kept of user accounts and system work. These record are kept in order to evaluate billing, as well as to calculate total resource utilization, availability, and demand response effectiveness. System administrators and managers measure effectiveness of scheduling policies, maintenance practices, and user allocations by this tools.

2.1.1 Workload

Workload is the amount of work to be done by system. To make decision about the workload assigning to the system, we have to define indicators. The indicators represent the use of system resources. Since the resources of computing systems are member of these four groups processor, memory, I/O and file, the indicators of workload are defined based on them [24, 25].

In high performance computing systems, load of processor and the length of its queue is the most common indicator for calculating workload. However, memory, bandwidth, inter process communication and disk have been proposed as workload indicators [26, 27].

2.1.2 Load Balancing Metrics

There are some metrics for load balancing mechanism that they are gathered as following [28, 29]:

I. ***Response Time:*** It is the amount of time taken to respond by a particular load balancing algorithm in a distributed system. This parameter should be minimized.

II. ***Throughput:*** It is used to determine the number of jobs whose execution has been completed. It should be high to improve the performance of the system.

III. ***Overhead:*** It determines the amount of overhead involved while implementing a load balancing algorithm. It is composed of overhead due to movement of tasks, inter-processor and inter-process communication. This should be minimized so that a load balancing technique can work efficiently.

IV. ***Fault Tolerance:*** It is the time to migrate the jobs or resources from one node to other. It should be minimized in order to enhance the performance of the system.

V. ***Resource Utilization:*** It is used to check the utilization of resources. It should be optimized for an efficient load balancing.

VI. ***Scalability:*** It is the ability of an algorithm to perform load balancing for a system with any finite number of nodes. This metric

2.2 Load Balancing Categorization

Nature of applications and computing platforms are two key parameters for designing efficient load balancing algorithms, we categorized algorithm parameters based on them.

Nature of application can be static or dynamic [30]. It’s clear that any applications need software and hardware resources to execute. Requirement of applications with static nature to resources is clear and measurable at compile time and it’s possible to allocate the resources to applications and distribute the load of the system before running the applications. In contrast to static applications, dynamic applications requirements to resources only be assessed at runtime. Actually, behavior of the dynamic applications is unpredictable and at runtime the processes of the application request new resources that were not considered previously or new process creates at the system or a priority of process execution change at runtime. In this situation, load balancer should be able to active at runtime and handle the new event and adjust the load of the system in order to better performance.

Computing platform is other effective parameter in designing load balancing algorithms that point to platforms characteristics. The underlying platform of computing system includes computer hardware, system software and networking. The main distinctions between different platforms are related to homogeneity and heterogeneity characteristics, open or close systems characteristics, scalable or non-scalable characteristics and structure of system characteristics.

The definition of a heterogeneous computing platforms depends to some extent on the application. The three main issues determining the classification are the hardware, the communication layer, and the software (operating system, compiler, compiler options). Any differences in these areas can potentially affect the behavior of the application. In heterogeneous computing platforms, it’s not possible to send equal workload to all nodes of the system same as homogeneous computing platforms [31].

Scalable systems are able to scale up or down in size, geographic and management at system life time duration [32]. Actually, scalability issue is defined in three dimensions, include number of users and/or processes (size scalability), maximum distance between nodes (geographical scalability) and number of administrative domains (administrative scalability). The three dimensions of scale affect distributed systems in many ways. Among the affected components are naming, authentication, authorization, accounting, communication, the use of remote resources, and the mechanisms by which users view the system. Scale affects reliability: as a system scales numerically, the likelihood that some host will be down increases; as it scales geographically, the likelihood that all hosts can communicate will decrease. Scale also affects performance: its numerical component affects the load on the servers and the amount of communication; its geographic component affects communication latency. Administrative complexity is also affected by scale:

administration becomes more difficult as changes become more frequent and as they require the interaction of different administrative entities, possibly with conflicting policies. Finally, scale affects heterogeneity: as the size of a system grows it. Therefore, we need an adaptive load balancer to be able consider new changes in this dynamic system.

The open systems let the nodes/elements to interact with nodes/elements from other open systems to work together to accomplish a job. The definition of open system definition has changed over time, but today open systems are usually considered to be systems that interoperate through open interfaces. In this situation, load balancer should be able to work independent of differences in underlying layers, closed systems do not work that way and initialized setting of the system don't change during life time of system [33].

Other important parameter of the platforms is related to their structure. System structure can be centralized, decentralized and distributed. Classification of a system as centralized, decentralized or distributed refers to communication and control organization, primarily. Based on the system structure, assignment of tasks to processes also can be classified as centralized, decentralized and distributed [34].

In centralized load balancer, master process holds a collection of tasks to be performed by the slave processes and when a task is completed, a slave process requests another. All slaves are the same (replicated worker), but specialized slaves capable of performing certain tasks are also possible [35].

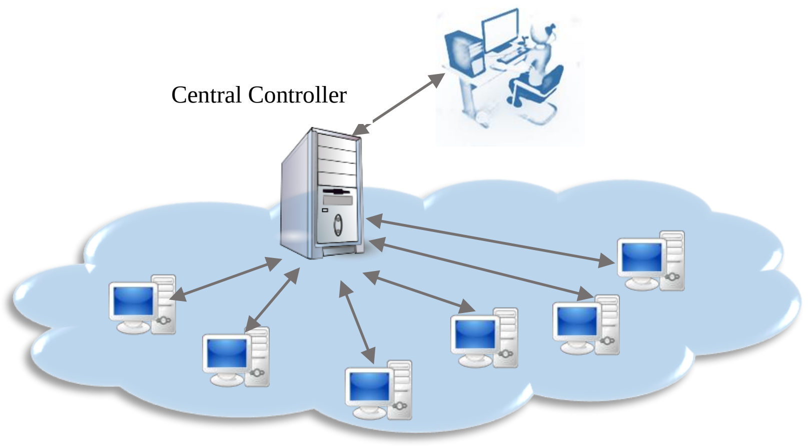


Figure 2.3: Centralized system diagram

Centralized systems have been in use for a long time such as in master and slaves based systems. The centralized systems directly control the operation of the individual nodes and flow of information from a single center. All individuals are directly dependent on the central manager to send and receive information, and to be commanded. The working diagram of a centralized system is shown in figure 2.3, in this system, individual units (represented by nodes in the figure 2.3), are directly controlled by the central manager. The individual nodes are forbidden to coordinate and work-together among themselves. Instead, each of them is obliged to follow the order from the center. The tasks are handed out from a centralized location and one dedicated master process directly controls each of a set of slave processes.

In decentralized system, management mechanism is decentralized in a hierarchical order such that there are middle tier powers between the central and local nodes [36]. In such decentralized system, one authority controls others directly below it and becomes controlled by the one directly above it. In doing so, the central authority can control the entire system. The working diagram of a decentralized system is shown in figure 2.4 too, the individual nodes are forbidden to coordinate and work-together among themselves. A decentralized system is also known as layered system or hierarchical system. In distributed load balancer, work pool is distributed among several master and each master controls one group of slaves.

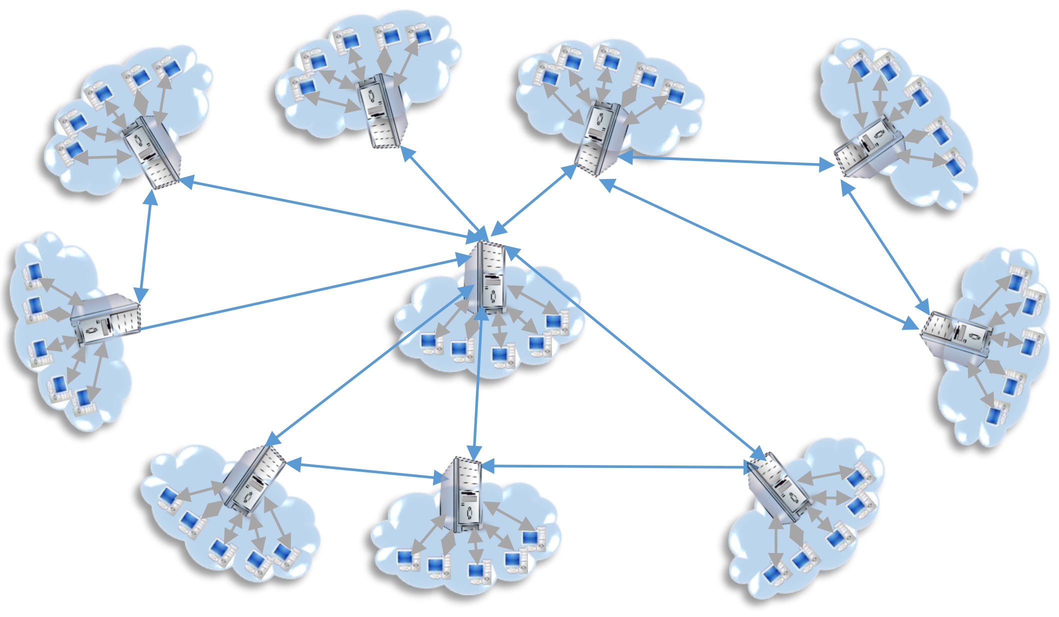


Figure 2.4: Decentralized System Diagram

A distributed system has no center manager and to not have to depend on a center manager for the functioning is the most prized asset of the distributed system. All nodes in a distributed system are networked on the basis of equality, independence, and cooperation. The lowest level nodes (authorities) can interact with their neighboring nodes using commonly agreed protocols, thereby building strong network that can be many times more resilient than centralized or decentralized systems [37].

The greatest advantage of this system is that its resilience increasing the number of the participants. The working diagram of a distributed system is shown in figure 2.5, where the individual nodes are sovereign to coordinate and work-together among themselves. A distributed system is also known as layer-less system or hierarchy-less system. A distributed system uses lateral (horizontal) protocols based on equality of relationship as opposed to a decentralized system, which uses hierarchical protocols where a higher node must always control the lower ones. Distributed systems permit independent nodes to make their own decisions.

In distributed load balancer, once tasks are (initially) distributed among processes (that moreover are able to generate new tasks), all processes can execute tasks from each other and tasks could be transferred by a receiver-initiated method or sender-initiated method. In receiver-initiated method a process that has only few or no tasks to perform requests tasks from other processes it selects (works well at high system loads). In sender-initiated method: a process with heavy load sends tasks to other processes it selects and that are willing to accept them (works well at light overall system loads).

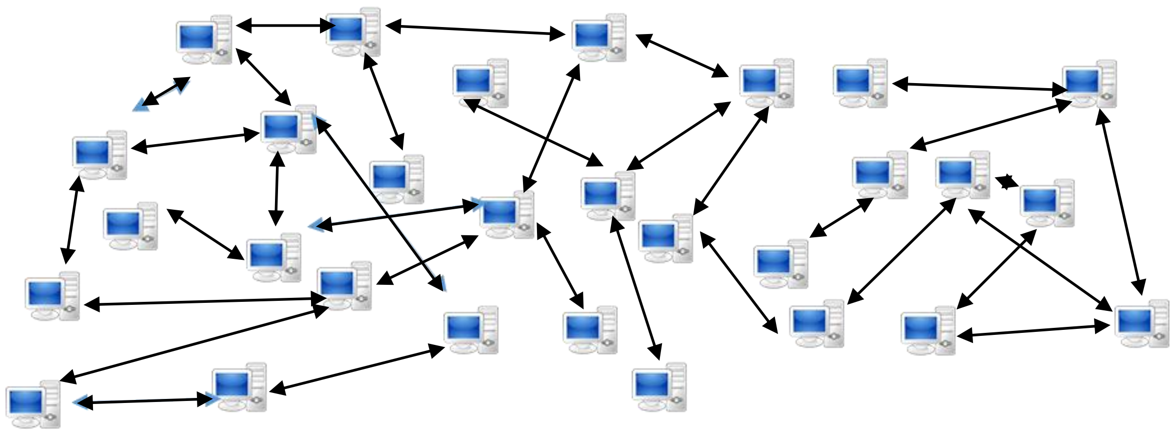


Figure 2.5: Distributed system diagram

Therefore, based on the two effective parameters in designing load balancing algorithms named nature of applications and computing platforms, we can conclude that we face two system situations, are mainly faced: predictable and unpredictable situations. Based on them, load balancing mechanism are categorized into two groups of mechanisms, static load balancing and dynamic load balancing [38]. In static load balancing mechanisms, the tasks are distributed between nodes at compile time and does not do any operation at runtime. The static load balancing mechanisms don’t let the new tasks execution at runtime duration. In contrast to static load balancing mechanisms, dynamic load balancing mechanisms do the process while tasks are in execution and accept new task execution during runtime. Dynamic load balancing is based on the redistribution of tasks among the processors during runtime. Dynamic load balancing have the potential of performing better than static strategies, they are inevitably more complex.

Advantages of dynamic load balancing policy include adaptiveness to changes in system parameters, such as delays in the communication channel, failure of nodes and the unknown characteristics of the incoming workloads. Dynamic load balancing generally reacts better to changes in the system state compared to the static methods and, as a result, have better performance. All load balancing policies discussed in this thesis are dynamic [39].

The policies that will be discussed in this thesis belong to the category of distributed load balancing, which are more robust, scalable and less susceptible to failure.

2.3 Dynamic Load Balancing

Dynamic load balancing techniques depend on recent system load information and determine the job assignments to server nodes at run time. In case of dynamic approach, the load balancing decisions are based on the current state of the system and hence workloads are allowed to shift dynamically from an overloaded node to an under-loaded node to get faster response from the server nodes. This ability to respond to variations in the system is the main advantage of the dynamic load balancing. But, since the load balancer has to continuously monitor the current system load on all the nodes, it becomes an extra overhead as monitoring consumes CPU cycles. Therefore, a proper decision must be taken when to invoke the monitoring operation by the load balancer [40, 41].

In this section, we discuss the various policies of dynamic load balancing and also discuss about different types of existing dynamic load balancing techniques along with their relative merits & demerits.

2.3.1 The Policies of Dynamic Load Balancing Algorithms

Different policies are used in load balancing algorithms. As mentioned in previous section standard load balancing manager is included five unites, Queuing, Scheduling, Resource Controlling, Monitoring and Accounting. Some of used policies in load balancing are introduces in these unites. Each of these unites should be designed based on the efficient policies to let us having efficient load balancing algorithms.

Information Policy in monitoring unit, Information policy is mainly responsible for collection of system state information. This policy specifies what kind of workload information to be collected, when it has to be collected and from where. In case of local state information, the state information of neighboring nodes is collected where as in case of global state information; state information of all the nodes in the system is collected for making scheduling decisions better.

Selection policies in scheduling/ rescheduling unite, selection policy deals with the selection of a task to be transferred. This policy defines the tasks that are to be transferred at the beginning time in scheduling unit or transferring the task from overloaded nodes to most idle nodes at the rescheduling phase. While transferring a task a basic criterion must be satisfied. This criterion is that the overhead incurred in the transfer of the task should be compensated by response time reduction.

Resource type policies in resource controlling unit, Resource policy deals with the specifying of a resource as a server node or receiver of workloads according to its availability status.

Location policies in resource controlling unit, Location policy uses the results of the resource type policy to find an appropriate partner for a server node or a receiver node. Location policy is mainly responsible for selecting the best server node among all the available nodes in the system. Resource availability and service availability are some of the factors which need to be considered while selecting a server node for workload execution.

Central Queue policies in queuing unit, in this Central Queue Algorithm, the new workload requests and the unfulfilled requests are stored in a cyclic FIFO queue on the main node. Each new work load request which arrives at the queue manager is inserted into the queue. Then, whenever a request for work load is received by the queue manager, it removes the first work load from the queue and sends it to the requesting node. If there are no available workloads in the queue then, the request is buffered, until a new work load is available. If a new work load arrives at the queue manager and at the same instance, there are un-serviced requests in the queue then, first such request is removed from the queue and the new work load is allocated to it. If any of the working node reaches the under loaded state then, the local load manager sends a request for a new work load to the central load manager. The central load manager responds the request immediately if a work load is available in the queue or queues the request until a new work load arrives.

Local Queue policies in queuing unit, This Local Queue policy, is featured with the support of dynamic process migration. The basic functionality of this policy is static assignment of all new processes with process migration initiated by a host when its load falls under threshold limit. This is a user-defined parameter of the algorithm. This parameter defines the minimal number of ready processes the load manager tries to allocate on each processor. At the initial stage, the new processes which are created on the main host are assigned on all under loaded hosts. The number of parallel activities created by the first parallel construct on the main host is usually sufficient for allocation on all remote hosts. From there after all the processes created on the main host and all other hosts are allocated locally. When the host gets under loaded, the local load manager tries to get several processes from remote hosts. It randomly picks up the local ready processes and sends requests with the number of local ready processes to remote load managers. When such requests are received by the load manager, it compares the local number of ready processes with that of the received number. If the local number is greater than the received number then, few running processes are shifted to the requester and a confirmation is sent with the number of processes transferred.

Queue Adjustment Policy (QAP) in queuing unit, in this Queue Adjustment Policy method, the scheduler which is the load manager is placed immediately after the queue. The algorithms which are following this policy attempts to balance the workloads in the queues of the nodes. When a work load arrives at node i, if the queue is empty, then the work load will be assigned to processor directly. If queue is not empty then, the work load is made to wait in the queue. The scheduler of node i periodically checks the queue lengths of other nodes that node i is interested. When there is any imbalance in the queue lengths i.e. few queues may be too long and few may be too short, the scheduler will make a decision on how many workloads in the queue should be transferred and where each of the workloads should be sent to. By doing the queue adjustment in this way, the load manager or the scheduler could balance the entire load in the system.

Rate Adjustment Policy (RAP) in scheduling unit, in this Rate Adjustment Policy method, the scheduler is immediately placed before the queue. When a work load arrives at node i, the scheduler makes a decision on where the work load should be sent, whether it needs to be sent to the queue of node i or to other nodes in the system. Once the work load enters the queue, it will be processed by the respective assigned node and will not be transferred to other nodes in the system.

Hybrid policy combination of queue and rate adjustment policy, in this Hybrid Policy method, the scheduler is allowed to adjust the incoming work load rate and also allowed to adjust the queue size of node i in some conditions. In some cases, when we use Rate Adjustment Policy method, the queue size may exceed a predefined threshold value due to which load imbalance may happen. When such situation happens, Queue Adjustment Policy method starts to work and guarantees that the workloads in the queues are balanced in the entire system. In this method, the rate adjustment can be considered as a coarse adjustment and the queue adjustment can be considered as a fine adjustment.

Some other used policies in load balancing algorithms are related to nature of load balancing such as triggering policy, Triggering policy specifies the appropriate time period to initiate a load balancing operation.

2.3.2 Centralized and Distributed Based Dynamic Load Balancing

As mentioned previously, the structure of the system has effective influence on designing the load balancing algorithms, because the location of decision making change by changing structure of the system. This section describes the different dynamic load balancing techniques based on the location of decision making, the information used for the decision making process, scalability factor, and the overhead of exchanging the profile information.

2.3.2.1 Centralized Dynamic Load Balancing

In this load balancing technique, a master node will have the responsibility of making the load balancing decision and the information used for the load balancing is obtained from the remaining slave nodes on either on demand basis or after a predefined fixed time interval. The load information may also be gathered only when there is any change in the system's current state. The advantage of this load balancing technique is that, since the load information is not send on arbitrarily, the inter process communication is reduced thereby avoiding network overhead. But, on the other hand, this technique has limited scalability feature.

2.3.2.2 Primary and Centralized Node Based Dynamic Load Balancing

In this method, either at the initial stage, processes can be stored in queue or the processes can be allocated to the nodes as they arrive. If processes are placed in queue then, each process is allocated one by one to the primary nodes of the load balancing system. During the course of time, in case if load imbalance happens in the system then, processes are migrated from heavily loaded nodes to lightly loaded nodes. Process migration makes a great impact on the network bandwidth and work load. In order to reduce the network traffic during process migration, nodes are grouped into clusters. At first, a lightly loaded node is checked in the same cluster, if a node is found then the process is transferred to the node which was found within the same cluster. If suitable node is not found within the same cluster then, it searches the nearby cluster and once a suitable node is found, the process transfer takes place.

In centralized node based load balancing technique, there are some situations where in, a heavily loaded node will not be able to find the lightly loaded node in its own cluster and due to network traffic; the node fails to search a node which would present in a remote cluster. It would be suitable if heavily loaded node finds a temporary node in its own cluster to handle the over load. So, in this Centralized approach, for every cluster a centralized node is provided and this node is not assigned with any work load initially. Whenever a primary node is over loaded, it first searches for the other lightly loaded primary nodes in its own cluster; if such primary node is available, the work load is transferred to the found node and the load is balanced in the system. If no other lightly loaded primary node is found within the same cluster then, the work load is assigned to the centralized node which is present within the same cluster. A Centralized node will have some better configuration structure when compared to other nodes in the cluster. The network traffic between centralized node and all other primary nodes are kept minimum in order to avoid network delays. Due to this reason, any overloaded node can easily reach the centralized node in case if it is heavily loaded and transfer the load easily.

2.3.2.3 Distributed Non-cooperative Dynamic Load Balancing

In this load balancing technique, the responsibility of distributing the work load is shared among all the working server nodes instead of using a master node. The current work load information is collected based on the on demand criteria. In case if any of the server node changes its current working state to overloaded state, then that specific node must distribute its current load information to all other nodes so that the current work load can be redistributed in order to balance the entire system load efficiently. This method provides moderate scalability as compare to the centralized method. But in case if any working node is overloaded then, since that overloaded node has to distribute its current load information to all other working nodes before rescheduling the system load, it may increase the network traffic due to inter process communication [44].

2.3.2.4 Distributed Cooperative Dynamic Load Balancing

In a cooperative form, nodes work together to achieve a global objective, e.g., to improve the system's overall response time. In distributed cooperative optimal dynamic load balancing techniques, unlike distributed non-cooperative dynamic scheduling techniques the responsibility of load balancing decision is scattered over all the workstations rather than master node. Further, in this case too load balancing information strategy is demand driven unlike the case of non-cooperative dynamic scheduling techniques with the exception of having average overhead during exchange of profile information. This technique does provide moderate scalability [45].

2.4 Load Balancing in Distributed Exascale Computing Systems

As computing nodes process tasks, some nodes may be overloaded while other nodes may run out of tasks, thereby becoming idle. To avoid such scenarios that defeat the purpose of cooperative computing, load balancing (LB) techniques are used so that the system resources are utilized optimally, and consequently the workload completion time is minimized. The most primary purpose of LB is to minimize the overall execution time of the initial workload. Since each node may have a different processing capacity, the workload has to be evenly distributed over all nodes according to their processing rates [46].

There has been extensive research in the development of effective load balancing policies and Among them, distributed dynamic load balancing (DLB) is a more useful approach for many distributed systems in practice [26].

The performance of DLB relies vitally on the accurate estimation of the load at each node. As nodes exchange information, they form their own individual estimates of the loads across the DCS based upon the received information, which can be local or global. DCSs may have large physical and/or logical distance among computing nodes, which result in large time delays in practice, due to the limitations existing in the communication environment of a DCS. Consequently, the load information that a node estimates about certain nodes, at the LB instant, is dated and may not accurately represent their current loads. The delays, in addition, may also be large in transferring tasks between nodes. Above time delay factors can seriously alter the expected performance of load balancing policies designed without taking into account such delays.

It is clear that large estimation errors of the loads (across the system) degrade the performance of DLB; it is less obvious that any disagreement among nodes in their estimates of the loads also degrades the performance. In most cases of interest, the inconsistency along the nodes in the network about their estimates of the loads can worsen the performance of the DLB compared to cases of similar average estimation error but with agreed upon estimates. Disagreement in the estimated loads results in conflict in LB decisions, which causes unnecessary task transfers among nodes. In summary, to improve the performance of DLB, the following inherent factors of the DCSs should be considered:

(i) The estimates of dynamic (local or global) load at each node.

(ii) The heterogeneity in the processing rates of the nodes.

(iii) The delays caused by the inherent constraints of the communication network.

We assume that the time for any load transfer across the network is assumed negligible, namely the communication delays of the network is relatively much lower compared to the execution time of one task. Therefore, the optimal load balancing instant should be at the instant when all nodes in the DCS reach the consensus on the estimates of the dynamic global load of the system. Clearly, there is a tradeoff between delaying the LB instant, in order to acquire more accurate estimates by receiving more information, and wasting computing resources as nodes may be kept idle due to large delays before a LB execution in distributed Exascale computing [27].

3 Proposed Load Balancing Model

As mentioned in previous sections, in load balancing with centralized architecture, there is a central node with responsibility of distributing the received load among nodes based on their execution capacity and bandwidth situation and the status of queue of jobs in the nodes.

In load balancing in distributed architecture, each node run load balancing mechanism itself and with cooperation of neighbor nodes, based on received status information from the neighbor nodes, distribute the extra load among the nodes. In fact, the local node make decision independently about balancing its load.

The suggested load balancing mechanism in this thesis is completely distributed because this mechanism is design to work in distributed Exascale computing systems and one of the important issue in this area is scalability of system.

In centralized architecture, scalability of system is completely limited by capacity of server node. Therefore, we choose distributed architecture,

Distributed systems based on status of nodes connections, are divided into two groups, named fully connected and partially connected distributed systems. In fully connected distributed systems, each node directly has access to all other nodes in the system while in partially connected distributed systems there is a concept named neighborhood and each node only has direct connection to its neighbors. In fact, real implemented distributed systems is partially connected network and the considered distributed system in this thesis is partially connected systems with support of scalability in size, geography and management.

In distributed systems, there are two strategies for distributing the loads, named sender-initiated strategy and receiver initiated strategy. In sender based strategy, the processor that is overloaded, and looking to send work to another initiates the transaction. In receiver based strategy, the under-loaded processor initiates the transaction by sending a message to other nodes. In some cases, receiver based algorithms are better and in some cases, depending on the algorithm and network topology, sender based may sometimes be better. However, the cost of task transfers under receiver-initiated policies is significantly greater than under sender-initiated policies, then sender-initiated policies provide uniformly better performance. In this thesis we used sender initiated strategy based on distributed platform.

There are also many algorithms for sender initiated load balancing. The single level load balancing method works by dividing a task into a larger number of subtasks, so that each processor can be made responsible for multiple subtasks. Task division of larger tasks into smaller tasks is handled by one processor called the “manager”. The manager generates the subtasks and distributes them to the requesting processors one at a time on demand. The manager must generate subtasks fast enough to keep all the other processors busy, otherwise a bottleneck is created and this scheme does not have good scalability.

A variation of the previous method of load balancing is multilevel load balancing. In this method, the processors are arranged in tress. The root processor of each tree is responsible for distributing subtasks to its child processors. If there is only one tree, this will act same as single level load balancing. However, when there are multiple trees, this will reduce the problem of all task management having to go through one processor. The processors that control each tree will also balance the load between them, so that load sharing is done across the entire network.

In this thesis we use another approach to load balancing named Sender Initiated Diffusion (SID) [47]. This strategy diffuses work to nearby neighbors to balance domains, and will eventually cause each processor load to be even. In this method, each processor acts independently, distributing some of its load to low-load neighbors. Balancing is done by each of the processors when it receives a load update from a neighbor stating that its load is below a threshold.

In the suggested load balancing mechanism, there are not any centralized operations, all operations such as gathering status information of nodes, distributing the load, gathering the results are done completely locally and distributed. It means all the nodes make decision about all the operations themselves and execute their load and balance it with cooperation of their neighbors and finally the total load of the system is balanced.

As mentioned before, load balancing mechanism is designed based on different concerns such as response time, utilization, throughput and etc. in this thesis our concern is decreasing response time/ total execution time by efficient use of resources and decreasing influence of communication delay.

In load balancing model with central architecture, the number of tasks in the system in time period Δt are related to mentioned parameters at below:

- The number of assigned tasks to computer i at the moment of t and we named it (L_i^{old})
- The number of assigned tasks to computer i at the moment of t+Δt and we named it (L_i^{new})
- The number of new assigned tasks to computer i in time period of Δt and we named it (J_i)
- The number of executed task in computer i in time period of Δt and we named it (C_i)
- The number of passed tasks from computer i to computer j in t_l^i time ($t < t_l^i < \Delta t$) that it is the result of load balancing mechanism and we show it by ($L_j(i)$).
- The number of passed task from computer j to computer i that they passed at the $t - \tau_{ji}^k$ time but because of communication delay these jobs arrived to computer i in time period of t+ Δt and we named it ($L_i(j)$)

Finally based on the introduced parameters, the following mathematical model is presented:

$$L_i^{new} = L_i^{old} - C_i + J_i - \sum_{j \neq i} L_j(i) + \sum_{j \neq i} L_i(j) \qquad \textbf{(3-1)}$$

In presented model in equation (3-1) the number of available tasks at the time period of t+Δt in node i are calculated. By calculating L_i^{new} , each node can estimate its status. In this thesis, each node by estimating its status by calculating L_i^{new} will be member of two following sets, I and I^c. As I is a set of nodes with positive load, and I^c is a set of nodes with negative load. The nodes with negative load, according to their computing power and volume of load that is intended to them to run, have more capacity to execute the tasks and it's possible to assign more task to them. On the other hand, the nodes with positive load has less computing power than the considered load for them and a portion of their load should be transferred to other nodes.

Essentially, this load balancing mechanism can be introduced for estimating $L_j(i)$ s, amount of transferred load from node with positive load, i, to nodes with negative load, j.

3.1 Important Issues in Load Balancing

In the previous sections, we talk about load balancing procedure in a system and the distributed way of this procedure. In this section, we introduces the major issues affecting the load balancing procedures such as amount of transferred load from the nodes with positive extra load to nodes with negative extra load, the start time of load balancing operation execution and the delay of transferring load from the nodes with positive load to nodes with negative load. Of course this is not just a distributed system and the central controller based systems are also effective.

I. Portion of Load Transferring

One of the important challenge of load balancing operation is portion of transferred load from nodes with positive load to nodes with negative load. In this section, we will describe this issue and proposed a model to specify this amount of load with goal of balancing the received load among all nodes and all nodes start to execute own portion and finally all nodes finish their execution at the same time.

II. Load Balancing Start Time

An important question at the issue of load balancing operations is “at what times the load balancing operation should be started?” While continuing execution of load balancing operations cause the response time of the assigned program increases, in designing a load balancing operation should use for suitable start time of load balancing operation. Different policies can be considered for load balancing start time. [48].

The load balancing policies based on number of load balancing operations execution and the node start load balancing execution are divided into different groups. From perspective of number of load balancing execution, we can divide them into two groups named One-time and reassignment. At the policy of one-time when the load is assigned to the nodes, it is not possible to change by other computers. But in Reassignment policy there is the possibility of revision in amount of assign load to each node.

In group of one-time policy, there is a policy named one-shot policy that we used in this thesis. In one-shot policy, upon entering the new load, load balancing operations can be performed only once. Actually this policy is based on one-time policy and in other hand sender initiated operation is used in one-shot policy. The implementation of such a policy seems quite logical. To illustrate further, consider system that the load balancing operation has been done once. Before entrance of new load, the system work normally based on before load balancing operation result and everything is according to plan, but as soon as entering new load to the system, the situation of the system is changed. That is why, it is necessary to once again perform load balancing operations. Therefore in this thesis, this policy for load balancing is used and this time is shown by t_b, that it is the moment of entering new load to the node.

III. Load Transferring Delay

Delay time of load transferring from the nodes with positive load to nodes with negative load should be acceptable and reasonable. If this amount of the transferring delay be more than the execution time of instructions in the node, load balancer should use suitable policy in balancing the load of the system while the total time of program execution be efficient. For example, consider a node that includes 200 task for execution and the average number of task execution in this node is 100 tasks per second. In this situation, the load balancer of the system makes decision to transfer 80 tasks to other node, therefore this node should execute 120 task that due to execution time of one task in this node the total time executing 120 task will be 1.2 task/s. Now suppose that transferring time of each task to destination node is 0.02 second.

In this situation, for transferring 80 tasks, 1.6 seconds is needed. Therefore, based on this load balancing policy, the source node (the node that distribute its extra load) don’t have any task for execution at least for 0.4 second. On the other hand, the destination node also can finish its tasks execution long before receiving these new tasks (transferred load). In both cases, part of the capacity of the system that could be used to execute commands will be lost. In this thesis, we introduce a parameter named "compensating factor" to transfer accurate portion of load by considering the delay time of transferring. This factor domain is $0 < k_l(r) \leq 1$ and we will describe its influence on our model in next section.

The parameter of k_{rl} causes the capacity of the system resource be never unusable. Due to the increasing the speed of processors, load transition delay in the real systems is very important issue and in this thesis we suggest a solution to solve this problem.

3.2 Centralized Load Balancing Mathematical Model

According to the mentioned description in previous section, the load balancing issue means specifying t_b and $L_j(i)$ in such way that minimize the response time of assigned tasks by efficient use of resources. As discussed, in relation to this issue, determining the number of assigned tasks and speed of the processors of nodes and taking into account the transition delay are very important.

For more appropriate description of this issue, consider a distributed system includes some nodes. Suppose that the i-th node has load to size of L_i and instruction execution in this node is λ_i instructions per second. With these assumptions, if the total assigned load to the system is equal to $\sum_k L_k$, the power of instruction executing in i-th computer than the power of entire system can be calculated by $\lambda_i \big/ \sum_k \lambda_k$. According to the mentioned assumptions and also the main objective of load balancing mechanism to efficient use of the resources in executing total assigned task to the system, it can be concluded that based on the relative power of the i-th node in entire system, this node should execute a portion of assigned load to the system. Therefore, we expect the amount of following portion of total load of the system be executed by i-th node:

$$\frac{\lambda_i}{\sum_k \lambda_k} \sum_k L_k \tag{3-2}$$

With this introduction, the extra load parameter in each node is equal to difference between the number of assigned tasks to l-th node and our expectation of amount of tasks that be executed by l-th node and we named this difference by L_l^{ex} and based on the mentioned definition we have:

$$L_l^{ex} = L_l - \frac{\lambda_l}{\sum_k \lambda_k} \sum_k L_k \tag{3-3}$$

In the other word, L_l^{ex} based on the computing power of i-th node than other nodes in the system gives an estimation of number of tasks (load) that should be transferred to other nodes in load balancing operation. The definition of L_l^{ex} in this thesis in addition to point numbers of tasks in each node, consider the speed of task execution in each node. It's clear that the total amount of L_l^{ex} s in the system is equal to zero, because:

$$\sum_l L_l^{ex} = \sum_l \left(L_l - \frac{\lambda_l}{\sum_k \lambda_k} \sum_k L_k \right) = \sum_l L_l - \sum_l \frac{\lambda_l}{\sum_k \lambda_k} \sum_k L_k = 0 \tag{3-4}$$

According to the mentioned definition about extra load, we can conclude that the nodes with negative load, relative to their computing power and volume of load that is intended to them to run, have more capacity to execute the tasks and it is possible to assign more task to them. On the other hand, the nodes with positive load has less computing power than the considered load for them and a portion of their load should be transferred to other nodes.

Based on L_l^{ex} parameter the mentioned two sets named I and I^c, as the sets with member of nodes with positive load, I, and the nodes with negative load, I^c are defined:

$$\mathrm{I} = \left\{ \mathrm{j} \mid L_j^{ex} < 0 \right\} \tag{3-5}$$

$$\mathrm{I}^c = \left\{ i \mid L_i^{ex} > 0 \right\} \tag{3-6}$$

Based on this definition, the load balancing problem can be defined as the transferring the extra load from the nodes with positive load to nodes with negative load. With this description, the amount of transferred load from the nodes such as r in I^c set ($r \in \mathrm{I}^c$) to the node such as l in I set ($l \in \mathrm{I}$), we named it $L_l(r)$, are calculate based on following equation:

$$L_l(r) = \left\lfloor p_l(r) L_r^{ex} \right\rfloor \tag{3-7}$$

In equation (3-7), $\lfloor . \rfloor$ is floor and ceiling operator and $p_l(r)$ factor is calculated based on the following equation:

$$p_l(r) = \frac{L_l^{ex}}{\sum_{k \in I} L_k^{ex}} \tag{3-8}$$

According the equation (3-8), we can conclude that $\sum_{l \in I} p_l(r) = 1$. According the result of the paper [49], by choosing this equation for $L_l(r)$, all nodes finish their load execution almost at the same time and the stand by time is at the minimum state. We use ‘almost’, because the amount of transferred load is calculated by floor and ceiling function and also load execution in each node is a based on a stochastic process and we can only have an average time for the execution time.

3.3 Distributed Load Balancing Mathematical Model

In this section, we introduce the suggested load balancing model with distributed approach. In this model, we define a factor named “Compensating factor” for calculating more accurate portion of load to transfer and also suggest some methods to calculate it.

One of the challenges in comparing different load balancing mechanism in the systems is long time of testing load execution. Therefore, we should find a solution to test different load balancing mechanisms and compare them with each other with minimum cost. According to stochastic nature of task execution in the computer, it seems that finding a statistical distribution to model the process of commands execution is essential. The behavior of a node to execute commands on average sequential execution of commands and performing an averaged action is available to model. But transient behavior of moment behavior of the node in executing the instructions should be modeled appropriately.

In telecommunication Engineering and especially on issues related to cellular communications, it’s usual to model the number of conversations conducted in a cell at a period of time, according to a population of cells, with Poisson distribution.

Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time and/or space if these events occur with a known average rate and independently of the time since the last event. The Poisson distribution can also be used for the number of events in other specified intervals such as distance, area or volume.

For instance, an individual keeping track of the amount of mail they receive each day may notice that they receive an average number of 4 letters per day. If receiving any particular piece of mail doesn't affect the arrival times of future pieces of mail, i.e., if pieces of mail from a wide range of sources arrive independently of one another, then a reasonable assumption is that the number of pieces of mail received per day obeys a Poisson distribution. Other examples that may follow a Poisson: the number of phone calls received by a call center per hour, the number of decay events per second from a radioactive source, or the number of pedicabs in queue in a particular street in a given hour of a day. A discrete random variable X is said to have a Poisson distribution with parameter λ > 0, if, for k = 0, 1, 2, ..., the probability mass function of X is given by:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \qquad \textbf{(3-9)}$$

Where:

e is Euler's number (e = 2.71828….)

k! is the factorial of k.

Poisson distribution probability function diagram shown in figure 3.1 to compare and showing the effectiveness of the use of Poisson distribution, a simple program for adding 1 + 1 is written and it is executed by a PC with a processor with core2dou 2.4GHz and 2 GB RAM. According to the results, the computer run x commands per second. This number is obtained by calculating he average speed of execution (λ) in 15 minutes time period.

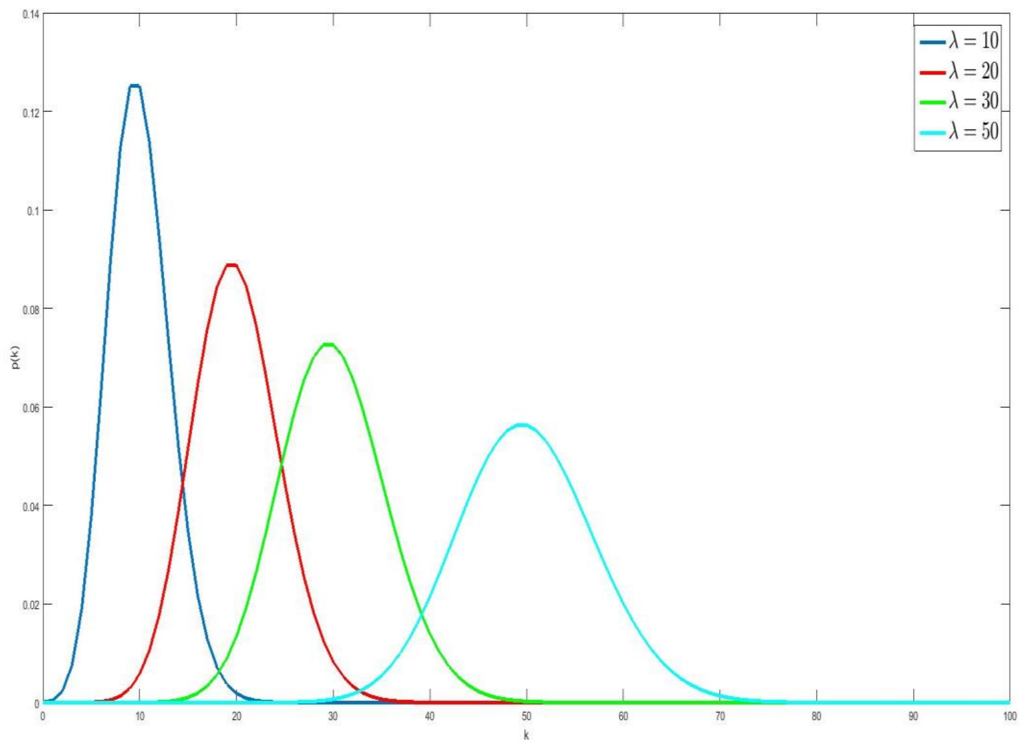


Figure 3.1: the Poisson distribution probability function

After estimating this number, based on some tests, number of executed commands in the same computer at intervals of 1 microsecond are measured and by repeating the tests in time period of 2 seconds, some charts are obtained from the moment behavior of executing the commands in that computer. Also by using Poisson function, some sample from Poisson distribution is produced. Figure 3.2 shows these two diagrams, as it can be seen, diagrams related to behavior of commands execution in the computer have a good agreement with the produced diagrams of Poisson distribution.

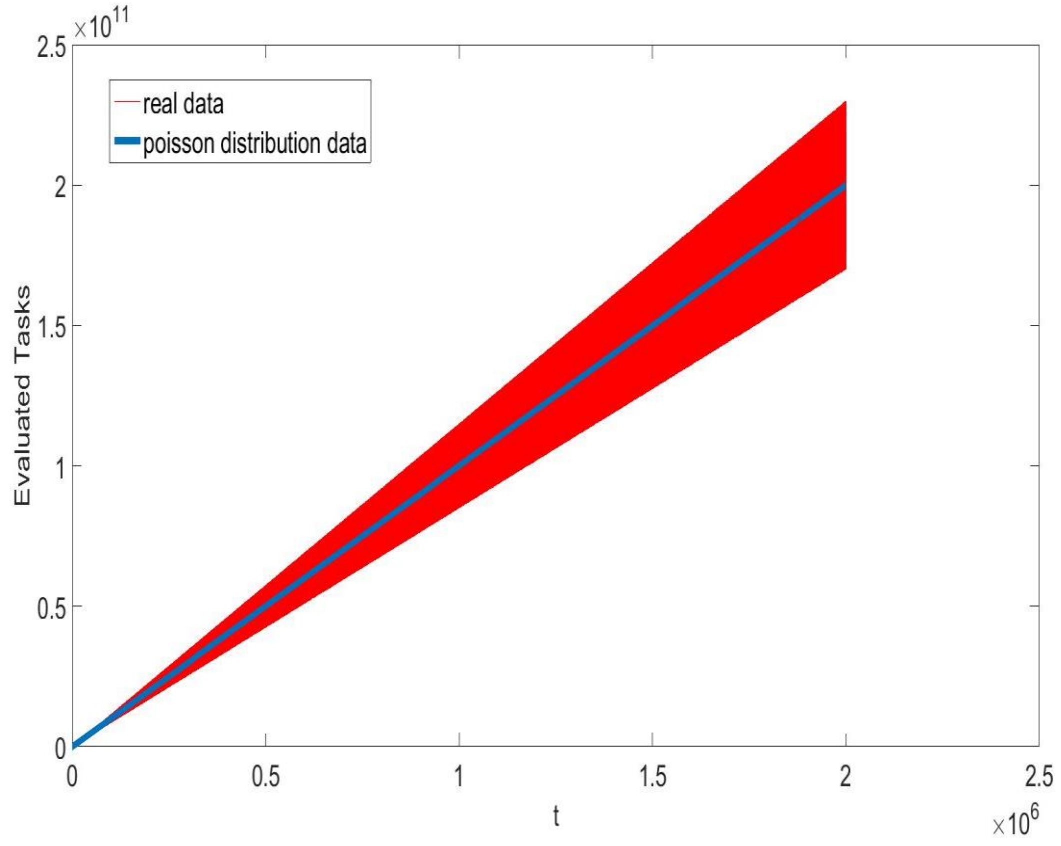


Figure 3.2: Comparison of numbers of executed commands in some real test (the red diagram) and numbers of executed commands in the same period of time by Poisson distribution

But to provide a suitable model for mathematical expression of load balancing issue, another parameter is needed to describe. As we know, the issue of load balancing is transferring additional loads from one node to another low load node to fast execution of total commands. It's clear that transferring the additional load from one node to another nodes imposes delay to the system. The communication delay according to high speed of commands execution in today's computers is a substantial amount that should not be neglected. For modeling the delay of sending commands from one node to another's, using an exponential distribution is accepted in engineering science. The exponential distribution (a.k.a. negative exponential distribution) is the probability distribution that describes the time between events in a Poisson process, i.e. a process in which events occur continuously and independently at a constant average rate. It is a particular case of the gamma distribution. It is the continuous analogue of the geometric distribution, and it has the key property of being memoryless. The probability density function of an exponential distribution is:

$$P(x = t) = \lambda e^{-\lambda x} \tag{3-10}$$

It should be noted that in the transition delay also transition line behavior can be modeled simply. But to model the behavior of these transitions, it’s possible to run some example of exponential distribution by “exprnd” command in MATLAB and model this delay.

3.4 Distributed Load Balancing

Since the subject of this thesis is modeling a distributed load balancer, we should consider the limitations of such system in the model accurately. Therefore, it is appropriate to model computer communication system with a graph. In this graph, each nodes of graph show a computing node and the edges show the connection of these computing nodes. For example in figure 3.3, the node number five is connected directly to node 4, node 1 and node 2.

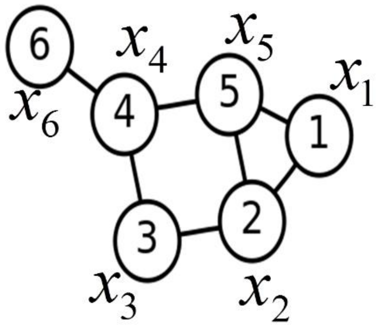


Figure 3.3: the graph of the distributed system model

For considering a distributed system the equation (3-1) should be amended. This relationship will be modified as follows:

$$L_i^{new} = L_i^{old} - C_i + J_i - \sum_{j \in N_i} L_j(i) + \sum_{j \in N_i} L_i(j) \qquad \textbf{(3-11)}$$

In addition, the load balancing equations are mentioned in section 3.2 should be modified. Because the all mentioned equations in that section are obtained based on this assumption that there is a central node in the system and all information about the nodes situation in the network; status of load in these nodes, are gathered by it. Therefore, this node can calculate $L_j(i)$ and send the result to other nodes.

For applying distributed feature and using one-shot mechanism, in the proposed load balancing mechanism in this thesis, each overloaded node act as a central node and start load balancing mechanism by using gathered information about status of neighbor nodes to manage this extra load. Status information of neighbor nodes means their current load status and also their speed up in executing commands. After passing time, the gathered information about the nodes status is not updated, in this situation, it's possible to estimate this information according to their average speed up in executing the commands and it's possible to estimate the status of them and use the result of this estimation in calculating extra load in the system. For example, if Δt seconds pass from receiving time of information of node i and the speed up of node i is λ_i, therefore, it's possible to estimate the load of node i by $L_i - \lambda_i \Delta t$ equation.

In this situation, for converting the central load balancer to distributed load balancer, first of all, we should add the index of the node with extra load in the equation to show that the calculated result are based on the view of this node.

For this reason, if assume that the extra load is entered in node i-th and this node should act as a central node and balance the load, the $L_l(r)$ in the equation (3-1) should change to $L_{il}(r)$ and L_r^{ex} should change to L_{ir}^{ex} .

For example, consider the figure 3.3 as the connection of distributed nodes. In this graph, assume that the extra load is entered to node 5 and the nodes of 1, 2 and 4 are their neighbor that nodes 1 and 2 have negative load and nodes 4, 5 have positive load. In this situation, our graph will be the figure 3.4 and the equation (3-7) and equation (3-8) will change.

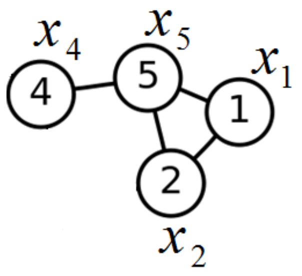


Figure 3.4: Equivalent graph for the case where the extra load is entering the fifth node.

If the nodes 1 and 2 have negative load and nodes 4 and 5 have positive load, our equation will be:

$$L_{51}(4) = \lfloor p_{51}(4) L_{54}^{ex} \rfloor$$

$$L_{51}(5) = \lfloor p_{51}(5) L_{55}^{ex} \rfloor$$

$$L_{52}(4) = \lfloor p_{52}(4) L_{54}^{ex} \rfloor$$

$$L_{52}(5) = \lfloor p_{52}(5) L_{55}^{ex} \rfloor$$

In which:

$$p_{51}(5) = p_{51}(4) = \frac{L_{51}^{ex}}{L_{51}^{ex} + L_{52}^{ex}}$$

$$p_{52}(5) = p_{52}(4) = \frac{L_{52}^{ex}}{L_{51}^{ex} + L_{52}^{ex}}$$

It should be noted that the first index in $L_{il}(r)$ and L_{il}^{ex} means the value of $L_l(r)$ and L_l^{ex} are from perspective of node i-th. This means that from perspective of node fifth, the system has a model in the form of figure 3.4 and not in the form of figure 3.3. Therefore, the equation (3-3) can be revised based on following equation:

$$L_{il}^{ex} = L_l - \frac{\lambda_l}{\sum_{k \in N_i} \lambda_k} \sum_{k \in N_i} L_k \qquad \textbf{(3-12)}$$

And also the equation (3-8) can be revised as shown in following:

$$p_{il}(r) = \frac{L_{il}^{ex}}{\sum_{k \in I_i} L_{ik}^{ex}} \qquad \textbf{(3-13)}$$

In which $\mathrm{I}_i = \{ l \mid l \in N_i, L_{il}^{ex} < 0 \}$.

3.5 Compensating Factor

As mentioned before, Due to the increased computational speed of modern computers, the delay caused by communication between computers becomes very important issue and should be taken into account in the calculations.

Before presentation of the proposed solution to the problem of delay between nodes, first of all we define the parameters used in continuing. Moreover, it is necessary to note that, since the topic of the thesis presents a solution for distributed load balancing operations, the defined parameters in the following have i index that show the value of the parameter from the perspective of i-th node.

- $\mathrm{I} = \left\{ i \mid L_i^{ex} < 0 \right\}$: set of nodes with negative load.
- $\mathrm{I}^c = \left\{ j \mid L_j^{ex} > 0 \right\}$: set of nodes with positive load.
- $\mathrm{I}_i = \left\{ \mathrm{l} \mid \mathrm{l} \in N_i, L_{il}^{ex} < 0 \right\}$: set of nodes with negative load from perspective of i-th node
- $\mathrm{I}_i^c = \left\{ \mathrm{r} \mid \mathrm{r} \in N_i, L_{ir}^{ex} > 0 \right\}$: set of nodes with positive load from perspective of i-th node.
- L_{ir}^{ex}, L_{il}^{ex} : the calculated extra load in r-th node and l-th node from perspective of i-th node
- $R = \left| \mathrm{I}_i \right|$: number of neighbor nodes to i-th node that have negative load.
- $L = \left| \mathrm{I}_i^c \right|$: number of neighbor nodes to i-th node that have positive load
- λ_{rl} : load transferring speed up from r-th node to l-th node.
- λ_r, λ_l : command execution speed up in r-th node and l-th node.

- T_{ir}, T_{il}: Command execution time in r-th node and l-th node from perspective of i-th node.
- T_r, T_l: command execution time in r-th node and l-th node
- $L_{il}(r)$: The amount of transferred load from i-th node to r-th node from perspective of i-th node.
- $L_l(r)$:the transferred load from r-th node to l-th node
- $P_{il}(r)$: Extra load transformation coefficient from r-th node to l-th node from perspective of i-th node.
- $P_l(r)$: Extra load transformation coefficient from r-th node to l-th node
- $K_{il}(r)$: Compensating Factor for transferring the load from r-th node ton l-th node from perspective of i-th node.
- $K_l(r)$: Compensating Factor for transferring the load from r-th node to l-th node.
- $t_{il}(r)$: The receiving time of the load transferred from r-th node to l-th node ($L_{il}(r)$) that it is calculated by i-th node.
- $t_l(r)$: The receiving time of the load transferred from r-th node to l-th node ($L_l(r)$)
- $t_{il}(r') = \{t_{il}(r) \mid r' \in \mathrm{I}_i, t_{il}(r'-1) < t_{il}(r')\}$: is set of $t_{il}(r)$ that it is just ascending.
- $L_{il}(r') = \{L_{il}(r) \mid r' \in \mathrm{I}_i, t_{il}(r'-1) < t_{il}(r')\}$: set of $L_{il}(r)$s that they are arranged according to $t_{il}(r')$s.
- $t_l(r') = \{t_l(r) \mid r' \in \mathrm{I}, t_l(r'-1) < t_l(r')\}$: set of $t_l(r)$s that it is just ascending.

- $L_l(r') = \{L_l(r) \mid r' \in \mathrm{I}, t_l(r'-1) < t_l(r')\}$: set of $L_l(r)$s that they are arranged according to $t_l(r')$s.

To demonstrate the effect of transmission delay time in the load balancing process and describing the model and logic of proposed approach for solving this problem, a factor named Compensating Factor is defined in this thesis. The factor is defined with approach of distributed system and from perspective of one of the nodes like i in the system that the node can be any nodes of the system.

Assume that the some external load entering to i-th node and this node wants to execute load balancing operation to compensate the effect of external load by using information of neighbor nodes status. It is obvious that all the nodes connected to the node i, can be divided into two groups I_i and I_i^c that they are defined:

$$\mathrm{I}_i = \{\mathrm{l} \mid \mathrm{l} \in N_i, L_{il}^{ex} < 0\} \quad \textbf{(3-14)}$$

$$\mathrm{I}_i^c = \{\mathrm{r} \mid \mathrm{r} \in N_i, L_{ir}^{ex} > 0\} \quad \textbf{(3-15)}$$

L_{ir}^{ex} and L_{il}^{ex} are the extra load of r-th and l-th nodes from perspective of i-th node. i-th node calculate the transferred load from r-th node to l-th node after calculating extra load by following equation:

$$L_{il}(r) = p_{il}(r) L_{ir}^{ex} \quad \textbf{(3-16)}$$

$$p_{il}(r) = \frac{L_{il}^{ex}}{\sum_{m \in \mathrm{I}_i} L_{im}^{ex}} \quad \textbf{(3-17)}$$

This amount of transferred load $L_{il}(r)$, with this assumption that the load transferred speed up from r-th node to l-th node is λ_{rl} , takes $t_{il}(r)$ second to reach to destination that in this equation:

$$t_{il}(r) = \frac{L_{il}(r)}{\lambda_{rl}} \tag{3-18}$$

For better description of the system, we sort the entered loads to i-th node ($L_{il}(r)$) based on the time of they get to the node ($t_l(r)$) and two following sets are defined:

$$L_{il}(r') = \{L_{il}(r) \mid r' \in \mathrm{I}_i , t_{il}(r'-1) < t_{il}(r')\} \tag{3-19}$$

$$t_{il}(r') = \{t_{il}(r) \mid r' \in \mathrm{I}, t_{il}(r'-1) < t_{il}(r')\} \tag{3-20}$$

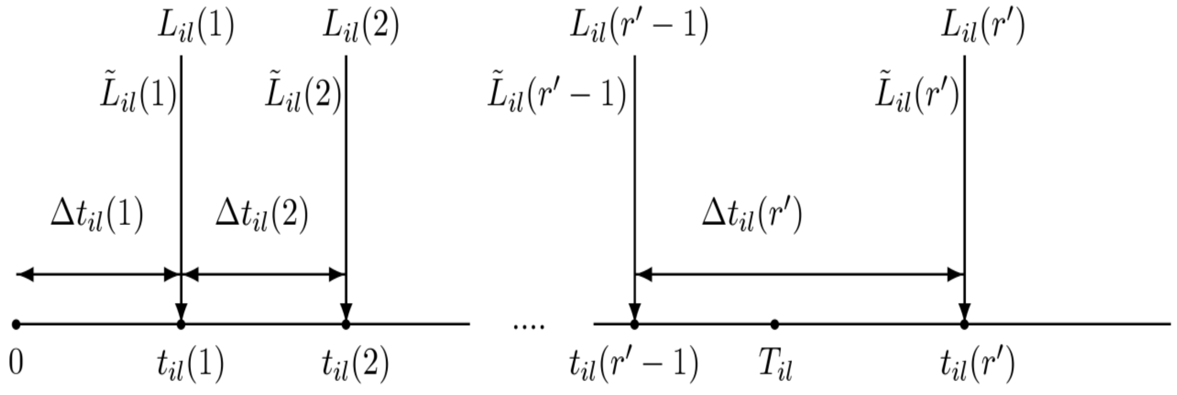


Figure 3.5: time model of load balancing in node i

The figure 3.5 shows the time model of events that happened in load balancing operation in i-th node from perspective of i-th node. The made time of the diagram is start time of load balancing operation and $\tilde{L}_{il}(r')$ show the amount of load of i-th node before reaching new load and we have:

$$\tilde{L}_{il}(r') = \left[\tilde{L}_{il}(r'-1) + L_{il}(r'-1) - \lambda_l \Delta t_{il}(r')\right]^+ \qquad \textbf{(3-21)}$$

In which:

$$\Delta t_{il}(1) = t_{il}(1) \qquad \textbf{(3-22)}$$

$$\Delta t_{il}(r') = t_{il}(r') - t_{il}(r'-1) \qquad \textbf{(3-23)}$$

Because of load transition delay, two events can be happened. First, the source node or the load transfer node finish its load before the load is transferred to the destination node. Happening of this event is not impossible, because, as mentioned in previous section, all the presented equations here are according to this condition that load of all nodes should be finished at the same time. Now, consider a situation that the source node send all the loads that should transfer based on load balancing operation.

Therefore, the time of complete execution of mentioned load is the end time of system operation. Now, if part of the load is on transition line, it means that at least one of the nodes didn’t receive its portion of total load. Therefore, although finishing load of the source node means all load of the system is finished, so this scenario is impossible.

The second event can be happened, assume that the l-th node finish its load execution before receiving $L_{il}(r')$ load from r'-th node. We show this finishing time by T_{il} and it is equal to:

$$T_{il} = t_{il}(r'-1) + \frac{\tilde{L}_{il}(r'-1) + L_{il}(r'-1)}{\lambda_l} \quad \textbf{(3-24)}$$

Actually, if the load of i-th node finish before receiving new load from r'-th node, part of the system resource capacity is lost. In other word, i-th node remain idle in time period of T_{il} to $t_{il}(r')$ while in this time period this node able to run $(t_{il}(r') - T_{il})\lambda_l$ commands. For better consideration of issue, the time period of $t_{il}(r'-1)$ to $t_{il}(r')$ is shown separately in figure 3.6.

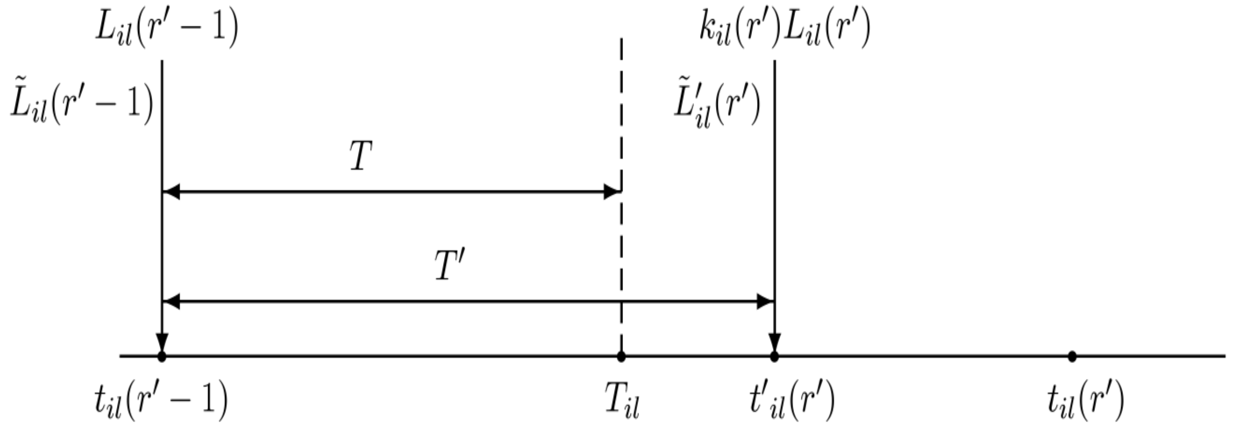


Figure 3.6: the time period between $t_{il}(r'-1)$ and $t_{il}(r')$

The important thing about the Compensating Factor is that this factor let us to minimize the idle time of the nodes in the system. As mentioned before, Compensating Factor is named by $k_{il}(r')$ a $k_{il}(r') \le 1$ and we change $L_{il}(r')$ to $k_{il}(r')L_{il}(r')$ and it causes the transfer time of load to l-th node decrease and arrive at the moment of $t'_{il}(r')$. Therefore, $k_{il}(r')$ factor is equal to:

$$k_{il}(r') = \frac{\lambda_{rl} t'_{il}(r')}{p_{il}(r')L^{ex}_{ir'}} = \frac{\lambda_{rl}\left[t_{il}(r'-1) + T'\right]}{p_{il}(r')L^{ex}_{ir'}} \quad \textbf{(3-25)}$$

Pay attention that the $k_{il}(r')$ parameter as mentioned above is a general form for choosing Compensating Factor. This is the way of choosing $t'_{il}(r')$ that specify the value of $k_{il}(r')$. In the other word, it's possible to gain new value for $k_{il}(r')$ by considering influence of different parameter in choosing $t'_{il}(r')$. In this thesis, three methods are proposed to choose $k_{il}(r')$ and present the analysis of the selected method. But because $p_{il}(r')L_{ir'}^{ex}$ is happened at $t_{il}(r') = t_{il}(r'-1) + \Delta t_{il}(r')$ time, we can write:

$$k_{il}(r') = \frac{\lambda_{r'l}\left[t_{il}(r'-1) + T'\right]}{\lambda_{r'l}\left[t_{il}(r'-1) + \Delta t_{il}(r')\right]} = \frac{t_{il}(r'-1) + T'}{t_{il}(r'-1) + \Delta t_{il}(r')} \quad \textbf{(3-26)}$$

3.6 Value of $k_{il}(r')$

It's clear that, the maximum value of $k_{il}(r')$ is one, in this situation, $L_{il}(r')$ will transfer that it is equal to the portion that was specified in last load balancing operation. Choosing larger than one value for $k_{il}(r')$, the node will be idle more, that in terms of load balancing operations are not justified. Because, as mentioned before, the load balancing operation should act in such a way that all nodes should finish their load at the same time.

According to equation (3-26), for $k_{il}(r') = 1$ we will have $T' = \Delta t_{il}(r')$. But the lower limit of $k_{il}(r')$ is the value that $t'_{il}(r') = T_{il}$. Choose a smaller amount of this limit has not justified. It should be noticed that the goal of choosing a value smaller than one for $k_{il}(r')$ is minimizing the idle time of l-th node and by choosing

$t'_{il}(r') = T_{il}$ its not necessary to minimize $k_{il}(r')$. In this situation, according to the equation (3-26), we have $T' = \Delta t_{il}(r')$. Therefore, approximate value of $k_{il}(r')$, T' and $t'_{il}(r')$ are calculated based on the following equations:

$$\frac{t_{il}(r'-1)+T}{t_{il}(r'-1)+\Delta t_{il}(r')} \le k_{il}(r') \le 1 \quad \textbf{(3-27)}$$

$$T \le T' \le \Delta t_{il}(r') \quad \textbf{(3-28)}$$

$$T_{il} \le t'_{il}(r') \le t_{il}(r') \quad \textbf{(3-29)}$$

Because of dependency of $k_{il}(r')$ ،T' and $t'_{il}(r')$, all are equivalent to each other. In continue, we will consider how to determine these values.

Choosing value for $k_{il}(r')$, For choosing a suitable value for $k_{il}(r')$, we should consider the impact of this selection and do reasonable selection. In continue, three methods should be proposed for choosing $k_{il}(r')$.

3.6.1 First Selection for K

As mentioned before, r'-th node transfer portion of its load to l-th node to solve problem of extra load and this amount of load is equal to $p_{il}(r')L^{ex}_{ir'}$.

By choosing this amount of load to transfer from r'-th node to l-th node and remain extra load of r'-th node will transfer to other nodes in set of I_i, the finish time of load execution in all nodes will be same. By selecting a value other than "one" for $k_{il}(r')$, amount of $k_{il}(r')p_{il}(r')L^{ex}_{ir'}$ load from $p_{il}(r')L^{ex}_{ir'}$ load transfer to l-th node and amount of $\left[1-k_{il}(r')\right]p_{il}(r')L_{ir'}^{ex}$ load remain in r’-th node that it causes r’-th node execute its load later that other nodes. This delay is equal to execution time of the mentioned amount of load remain in r’-th node and this time is depended to load execution speed up in r’-th node that we show it by $\lambda_{r'}$ and this time is equal to:

$$\frac{\left[1-k_{il}(r')\right]p_{il}(r')L_{ir'}^{ex}}{\lambda_{r'}}=\frac{p_{il}(r')L_{ir'}^{ex}}{\lambda_{r'}}-\frac{k_{il}(r')p_{il}(r')L_{ir'}^{ex}}{\lambda_{r'}}$$

$$=\frac{\lambda_{r'l}}{\lambda_{r'}}t_{il}(r')-\frac{\lambda_{r'l}}{\lambda_{r'}}t'_{il}(r')=\frac{\lambda_{r'l}}{\lambda_{r'}}\left[t_{il}(r')-t'_{il}(r')\right]$$

$$=\frac{\lambda_{r'l}}{\lambda_{r'}}\left[t_{il}(r'-1)+\Delta t_{il}(r')-t_{il}(r'-1)-T'\right] \qquad \textbf{(3-30)}$$

$$=\frac{\lambda_{r'l}}{\lambda_{r'}}\left[\Delta t_{il}(r')-T'\right]$$

For first selection in determining $k_{il}(r')$, we suggest to put equal the idle time of l-th node with idle time of r’-th node. Since the idle time of l-th node is equal to $T'-T$, based on the figure 3.4, and idle time of r’-th node based on the equation (3-30) is written in the following:

$$\frac{\lambda_{r'l}}{\lambda_{r'}}\left[\Delta t_{il}(r')-T'\right]=T'-T \qquad \textbf{(3-31)}$$

By simplifying the equation (3-31), we have:

$$\frac{\lambda_{r''}}{\lambda_{r'}}\Delta t_{il}(r')+T = T'\left(1+\frac{\lambda_{r''}}{\lambda_{r'}}\right)$$

$$\frac{\lambda_{r''}}{\lambda_{r'}}\Delta t_{il}(r')+T = T'\left(\frac{\lambda_{r'}+\lambda_{r''}}{\lambda_{r'}}\right)$$

That in this situation we will have:

$$T' = \frac{\lambda_{r''}}{\lambda_{r'}+\lambda_{r''}}\Delta t_{il}(r') + \frac{\lambda_{r'}}{\lambda_{r'}+\lambda_{r''}}T \qquad \textbf{(3-32)}$$

Value of $k_{il}(r')$ based on both T' and equation (3-26) will be:

$$k_{il}(r') = \frac{t_{il}(r'-1) + \frac{\lambda_{r''}}{\lambda_{r'}+\lambda_{r''}}\Delta t_{il}(r') + \frac{\lambda_{r'}}{\lambda_{r'}+\lambda_{r''}}T}{t_{il}(r'-1)+\Delta t_{il}(r')}$$

$$= \frac{\frac{\left[\lambda_{r'}+\lambda_{r''}\right]t_{il}(r'-1)}{\lambda_{r'}+\lambda_{r''}} + \frac{\lambda_{r''}}{\lambda_{r'}+\lambda_{r''}}\Delta t_{il}(r') + \frac{\lambda_{r'}}{\lambda_{r'}+\lambda_{r''}}T}{t_{il}(r'-1)+\Delta t_{il}(r')}$$

$$= \frac{\frac{\lambda_{r''}}{\lambda_{r'}+\lambda_{r''}}\left[t_{il}(r'-1)+\Delta t_{il}(r')\right] + \frac{\lambda_{r'}}{\lambda_{r'}+\lambda_{r''}}\left[t_{il}(r'-1)+T\right]}{t_{il}(r'-1)+\Delta t_{il}(r')}$$

Finally we will have:

$$k_{il}(r') = \frac{\lambda_{r''}}{\lambda_{r'}+\lambda_{r''}} + \frac{\lambda_{r'}}{\lambda_{r'}+\lambda_{r''}}\left[\frac{t_{il}(r'-1)+T}{t_{il}(r'-1)+\Delta t_{il}(r')}\right] \qquad \textbf{(3-33)}$$

In equation (3-32), if the speed up of load transition from r'-th node to l-th node goes to infinity, it means $\lambda_{r'l} \to +\infty$, then we will have $k_{il}(r') \to 1$ and when $\lambda_{r'l} \to 0$ then we will have $k_{il}(r') \to \frac{t_{il}(r'-1)+T}{t_{il}(r'-1)+\Delta t_{il}(r')}$ that it is equal to the same period intended for $k_{il}(r') \to 1$ in equation (3-27).

3.6.2 Second Selection for K

The important point is not intended at above equations and that is related to determination of T'. For determining T', idle time of l-th node and additional work of r'-th node have same priority.

By selecting T' in such a way we select in first suggestion, the following events should be happened:

- In l-th node in time period of T' to T for $T'-T$ second the l-th node is remain idle.
- l-th node finish its load execution with $T'-T$ second delay because it has been idle for $T'-T$. On the other hand, this node receive $k_{il}(r')p_{il}(r')L^{ex}_{ir'}$ load instead of $p_{il}(r')L^{ex}_{ir'}$, so it will finish its load for $\left[1-k_{il}(r')\right]p_{il}(r')L^{ex}_{ir'}/\lambda_l$ seconds sooner. As a result, l-th node will finish its load based on the following equation:

$$T'-T-\frac{\left[1-k_{il}(r')\right]}{\lambda_l}p_{il}(r')L^{ex}_{ir'} \qquad \textbf{(3-34)}$$

By using equation (3-30), we can write:

$$T'-T-\frac{\left[1-k_{il}(r')\right]}{\lambda_l}p_{il}(r')L^{ex}_{ir'}=T'-T-\frac{\lambda_{r'l}}{\lambda_l}\left[\Delta t_{il}(r')-T'\right] \qquad \textbf{(3-35)}$$

- r’-th node will finish its load $\frac{[1-k_{il}(r')]}{\lambda_{r'}} p_{il}(r')L^{ex}_{ir'}$ seconds later, because

instead of $p_{il}(r')L^{ex}_{ir'}$ load, it transfers $k_{il}(r')p_{il}(r')L^{ex}_{ir'}$ load and based on the equation (3-30) we can write:

$$\frac{[1-k_{il}(r')]\,p_{il}(r')L^{ex}_{ir'}}{\lambda_{r'}} = \frac{\lambda_{r'l}}{\lambda_{r'}}\left[\Delta t_{il}(r') - T'\right] \qquad \textbf{(3-36)}$$

The second solution for selecting k is based on minimizing all these idle time. For this purpose, a positive cost function of all idle times is defined and by derivative of this function with respect to T', we will calculate a value for T' that minimize this function. Based on this solution, we can define A as follows:

$$A = (T' - T)^2 + \left(T' - T - \frac{\lambda_{r'l}}{\lambda_l}\left[\Delta t_{il}(r') - T'\right]\right)^2 + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\left[\Delta t_{il}(r') - T'\right]\right)^2 \qquad \textbf{(3-37)}$$

By calculating the deviation of A with respect to T', we will have:

$$\frac{dA}{dT'} = 2(T' - T) + 2\left(1+\frac{\lambda_{r'l}}{\lambda_l}\right)\left(T' - T - \frac{\lambda_{r'l}}{\lambda_l}\left[\Delta t_{il}(r') - T'\right]\right) - 2\left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)\left(\frac{\lambda_{r'l}}{\lambda_{r'}}\left[\Delta t_{il}(r') - T'\right]\right)$$

By putting the above equation equal to zero, we will have:

$$T' + \left(1+\frac{\lambda_{r'l}}{\lambda_l}\right)^2 T' + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2 T' = T + \left(1+\frac{\lambda_{r'l}}{\lambda_l}\right)T + \left(1+\frac{\lambda_{r'l}}{\lambda_l}\right)\frac{\lambda_{r'l}}{\lambda_l}\Delta t_{il}(r') + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2 \Delta t_{il}(r')$$

$$\left[2+\left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + 2\left(\frac{\lambda_{r'l}}{\lambda_l}\right) + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2\right]T' = \left(2+\frac{\lambda_{r'l}}{\lambda_l}\right)T + \left[\frac{\lambda_{r'l}}{\lambda_l} + \left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2\right]\Delta t_{il}(r')$$

That T' will be equal to:

$$T' = \frac{\left(2+\frac{\lambda_{r'l}}{\lambda_l}\right)T + \left[\frac{\lambda_{r'l}}{\lambda_l} + \left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2\right]\Delta t_{il}(r')}{2+\left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + 2\left(\frac{\lambda_{r'l}}{\lambda_l}\right) + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2} \qquad \textbf{(3-38)}$$

$k_{il}(r')$ value based on this value for T' and by using equation (3-26) will be:

$$k_{il}(r') = \frac{t_{il}(r'-1) + \frac{\left(2+\frac{\lambda_{r'l}}{\lambda_l}\right)T + \left[\frac{\lambda_{r'l}}{\lambda_l} + \left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2\right]\Delta t_{il}(r')}{2+\left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + 2\left(\frac{\lambda_{r'l}}{\lambda_l}\right) + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2}}{t_{il}(r'-1) + \Delta t_{il}(r')}$$

$$k_{il}(r') = \frac{\frac{\left(2+\frac{\lambda_{r'l}}{\lambda_l}\right)\left[t_{il}(r'-1)+T\right] + \left[\frac{\lambda_{r'l}}{\lambda_l} + \left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2\right]\left[t_{il}(r'-1)+\Delta t_{il}(r')\right]}{2+\left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + 2\left(\frac{\lambda_{r'l}}{\lambda_l}\right) + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2}}{t_{il}(r'-1) + \Delta t_{il}(r')}$$

$$k_{il}(r') = \frac{\frac{\lambda_{r'l}}{\lambda_l} + \left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2}{2+\left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + 2\left(\frac{\lambda_{r'l}}{\lambda_l}\right) + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2}$$

$$+\frac{2+\frac{\lambda_{r'l}}{\lambda_l}}{2+\left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + 2\left(\frac{\lambda_{r'l}}{\lambda_l}\right) + \left(\frac{\lambda_{r'l}}{\lambda_{r'}}\right)^2}\left[\frac{t_{il}(r'-1)+T}{t_{il}(r'-1)+\Delta t_{il}(r')}\right] \qquad \textbf{(3-39)}$$

In equation (3-38), if the load transition speed up from r'-th node to l-th node go to infinity, it means $\lambda_{r'l} \to +\infty$, then we will have $T' \to \Delta t_{il}(r')$ and whenever $\lambda_{r'l} \to 0$ then we will have $T' \to T$. Actually, T' is in this interval $T \le T' \le \Delta t_{il}(r')$, that it's the considered interval for $T' \to \Delta t_{il}(r')$ in equation (3-28). In addition, according to equation (3-39), if the load transition speed up from r'-th node to l-th node go to infinity, it means $\lambda_{r'l} \to +\infty$, then $k_{il}(r') \to 1$ and if $\lambda_{r'l} \to 0$ then we will have $k_{il}(r') \to \frac{t_{il}(r'-1)+T}{t_{il}(r'-1)+\Delta t_{il}(r')}$ that its equal to the considered interval for $k_{il}(r') \to 1$ in equation (3-27).

3.6.3 Third Selection for K

In previous selection for T', it was intended to minimize total idle time of nodes. But the important point that should be considered is that the idle time is not only important parameter. Actually, the most important point is number of commands that the nodes can execute in this idle time. Therefore:

- In l-th node in time period of T and T', $(T'-T)\lambda_l$ commands can be executed
- l-th node will finish its work for $T'-T$ second later because as much as this time period, the l-th node was idle. On the other hand, l-th node receive $k_{il}(r')p_{il}(r')L_{ir'}^{ex}$ load instead of $p_{il}(r')L_{ir'}^{ex}$ load, for $[1-k_{il}(r')]p_{il}(r')L_{ir'}^{ex}/\lambda_l$ seconds finish its work sooner. Therefore, totally l-th node can execute as much the following load:

$$(T'-T)\lambda_l - [1-k_{il}(r')]p_{il}(r')L_{ir'}^{ex} \qquad \textbf{(3-40)}$$

By using the equation (3.29), we will have:

$$\left(T'-T\right)\lambda_l-\left[1-k_{il}(r')\right]p_{il}(r')L_{ir'}^{ex}=\left(T'-T\right)\lambda_l-\left[\Delta t_{il}(r')-T'\right]\lambda_{r'l} \quad \textbf{(3-41)}$$

- r'-th node transfer $k_{il}(r')p_{il}(r')L_{ir'}^{ex}$ load instead of $p_{il}(r')L_{ir'}^{ex}$ load, so $\left[1-k_{il}(r')\right]p_{il}(r')L_{ir'}^{ex}$ load can be executed by r'-th node and we will have the following equation:

$$\frac{\left[1-k_{il}(r')\right]p_{il}(r')L_{ir'}^{ex}}{\lambda_{r'}}=\frac{\lambda_{r'l}}{\lambda_{r'}}\left[\Delta t_{il}(r')-T'\right] \quad \textbf{(3-42)}$$

By defining B as following:

$$B=\left(T'-T\right)^2\lambda_l^2+\left[\left(T'-T\right)\lambda_l-\left[\Delta t_{il}(r')-T'\right]\lambda_{r'l}\right]^2+\left(\left[\Delta t_{il}(r')-T'\right]\lambda_{r'l}\right)^2 \quad \textbf{(3-43)}$$

By calculating the deviation of B with respect to T', we will have:

$$\frac{dB}{dT'}=2\left(T'-T\right)\lambda_l^2+2\left(\lambda_l+\lambda_{r'l}\right)\left(\left[T'-T\right]\lambda_l-\left[\Delta t_{il}(r')-T'\right]\lambda_{r'l}\right)-2\lambda_{r'l}^2\left[\Delta t_{il}(r')-T'\right]$$

With equating the above equation to zero, we will have:

$$\left(2\lambda_l^2+2\lambda_{r'l}\lambda_l+2\lambda_{r'l}^2\right)T'=\left(2\lambda_l^2+\lambda_{r'l}\lambda_l\right)T+\left(2\lambda_{r'l}^2+\lambda_{r'l}\lambda_l\right)\Delta t_{il}(r')$$

In this situation T' will be:

$$T'=\frac{\left(2\lambda_l^2+\lambda_{r'l}\lambda_l\right)T+\left(2\lambda_{r'l}^2+\lambda_{r'l}\lambda_l\right)\Delta t_{il}(r')}{2\lambda_l^2+2\lambda_{r'l}\lambda_l+2\lambda_{r'l}^2} \quad \textbf{(3-44)}$$

Value of $k_{il}(r')$ commensurate with this amount of T' and by using equation (3-26), will be:

$$k_{il}(r')=\frac{t_{il}(r'-1)+\dfrac{\left(2\lambda_l^2+\lambda_{r'l}\lambda_l\right)T+\left(2\lambda_{r'l}^2+\lambda_{r'l}\lambda_l\right)\Delta t_{il}(r')}{2\lambda_l^2+2\lambda_{r'l}\lambda_l+2\lambda_{r'l}^2}}{t_{il}(r'-1)+\Delta t_{il}(r')}$$

$$k_{il}(r')=\frac{\dfrac{\left(2\lambda_l^2+\lambda_{r1}\lambda_l\right)\left(t_{il}(r'-1)+T\right)+\left(2\lambda_{r1}^2+\lambda_{r1}\lambda_l\right)\left[t_{il}(r'-1)+\Delta t_{il}(r')\right]}{2\lambda_l^2+2\lambda_{r1}\lambda_l+2\lambda_{r1}^2}}{t_{il}(r'-1)+\Delta t_{il}(r')}$$

$$k_{il}(r')=\frac{2\lambda_{r1}^2+\lambda_{r1}\lambda_l}{2\lambda_l^2+2\lambda_{r1}\lambda_l+2\lambda_{r1}^2}+\left(\frac{2\lambda_l^2+\lambda_{r1}\lambda_l}{2\lambda_l^2+2\lambda_{r1}\lambda_l+2\lambda_{r1}^2}\right)\frac{t_{il}(r'-1)+T}{t_{il}(r'-1)+\Delta t_{il}(r')} \quad \textbf{(3-45)}$$

As you can see in equation (3-44), if the load transfer speed up from r'-th node to l-th node go to infinity, it means $\lambda_{r1}\rightarrow+\infty$, then we will have $T'\rightarrow\Delta t_{il}(r')$, and if the mentioned speed up go to zero, it means, $\lambda_{r1}\rightarrow 0$, then we will have $T'\rightarrow T$. Actually, T' is in this time period: $T\leq T'\leq\Delta t_{il}(r')$, that this period of time specify the same considered period of time for $T'\rightarrow\Delta t_{il}(r')$ in equation (3-28). In addition, according to equation (3-45), if the load transferring speed up go to infinity, it means $\lambda_{r1}\rightarrow+\infty$, then $k_{il}(r')\rightarrow 1$ and if $\lambda_{r1}\rightarrow 0$ then $k_{il}(r')\rightarrow\frac{t_{il}(r'-1)+T}{t_{il}(r'-1)+\Delta t_{il}(r')}$ that it's same as the considered time period for $k_{il}(r')\rightarrow 1$ in equation (3-27).

3.6.4 Comparing Second and Third Selection Methods

As described before, the second and third methods have the same fundamental. One of the similarity of second and third methods is if the speed up of source and destination nodes are equal (r' , l) and $\lambda_{r'}=\lambda_l$ then we will have in equation (3-39):

$$k_{il}(r') = \frac{\frac{\lambda_{r'l}}{\lambda_l} + \left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + \left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2}{2 + \left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + 2\left(\frac{\lambda_{r'l}}{\lambda_l}\right) + \left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2} + \frac{2 + \frac{\lambda_{r'l}}{\lambda_l}}{2 + \left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2 + 2\left(\frac{\lambda_{r'l}}{\lambda_l}\right) + \left(\frac{\lambda_{r'l}}{\lambda_l}\right)^2}\left[\frac{t_{il}(r'-1)+T}{t_{il}(r'-1)+\Delta t_{il}(r')}\right] \tag{3-46}$$

By multiplying the numerator and denominator of above equation to λ_l, the mentioned equation will be same as equation (3-45).

One of the more important advantage of the three proposed methods for calculating $k_{il}(r')$, in addition, improvement of the efficiency of the system that we will describe in next section, is related to accurate selected value of $k_{il}(r')$. All selected value for $k_{il}(r')$ in three methods are based on a close and specified formula and it causes the computing complexity of it be low.

4 Evaluation of the Proposed Model

We evaluate the performance of our proposed algorithms using computer simulations which is coded as a toolbox for MATLAB software. We consider a computer network composed of multi system as a connected graph. As mentioned earlier we assume that the distributions of task process speed in each node and the communication delay between nodes are Poisson and exponential respectively.

Load balancing with one-shot policy in a multi computer network depends on different parameters including task process speed in each node, delay of load transferring between two nodes, the initial load of each node, network graph model that defines the connection between nodes, external load insertion time instant and the node facing the external node.

In this chapter, we define a simulation scenario with controlled variation of mentioned parameters to evaluate the effect of load-transfer delay and to evaluate the performance of proposed algorithms.

We consider a network with 6 nodes, each having 300 initial tasks and all with the same processor speed. Considering one-shot policy, we assume external load as 1000 task implied to node 1. The network topology is assumed to be as follows:

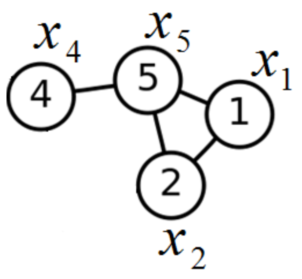


Figure 5.1: the network topology of nodes

First of all, we assume that all nodes are capable to perform 500 tasks per second and transfer 100 tasks from one to another per second. We assume that the insertion time of external load to system is at t=0.1. If we do not use the load balancing, the execution process of tasks in the network is as follows:

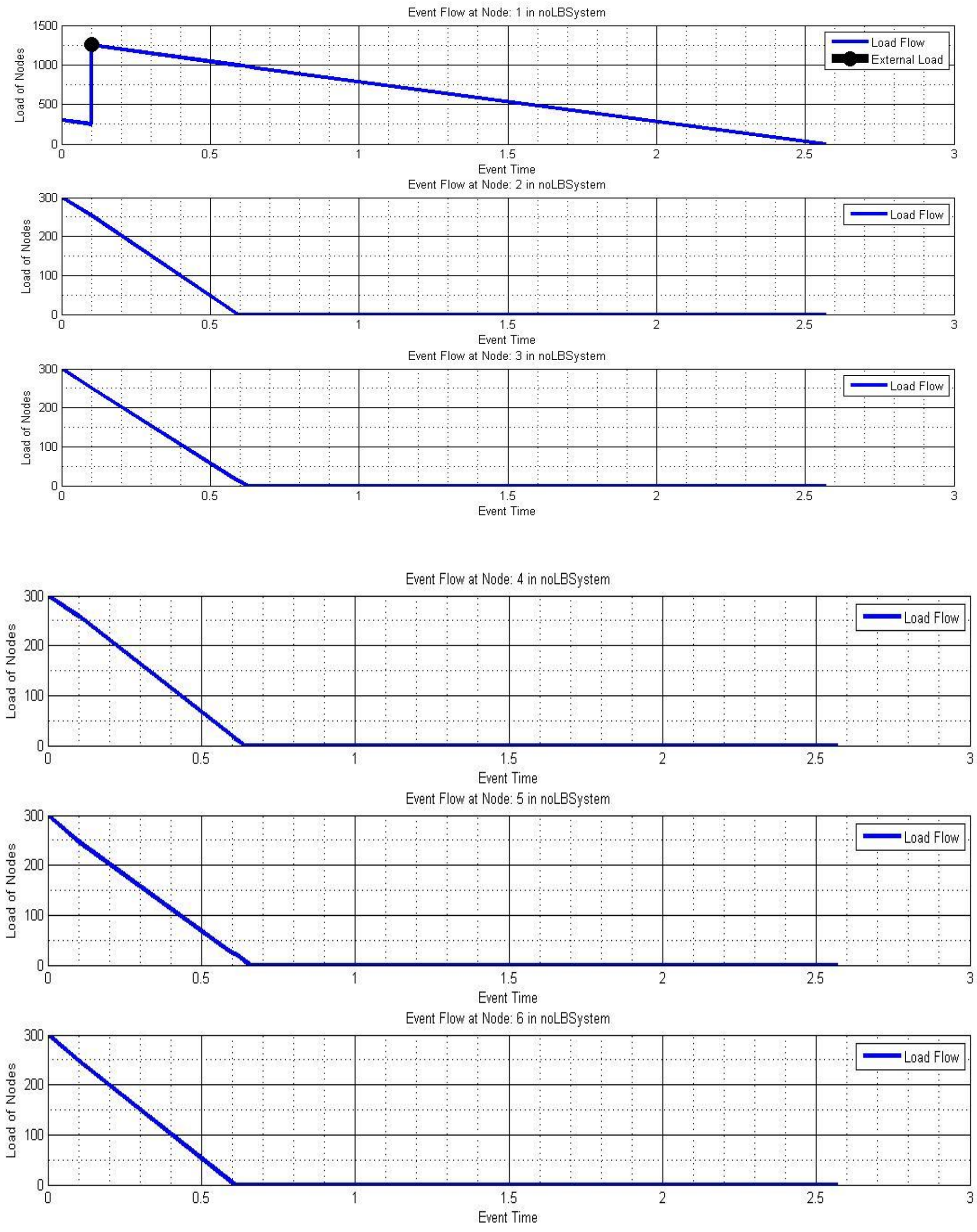
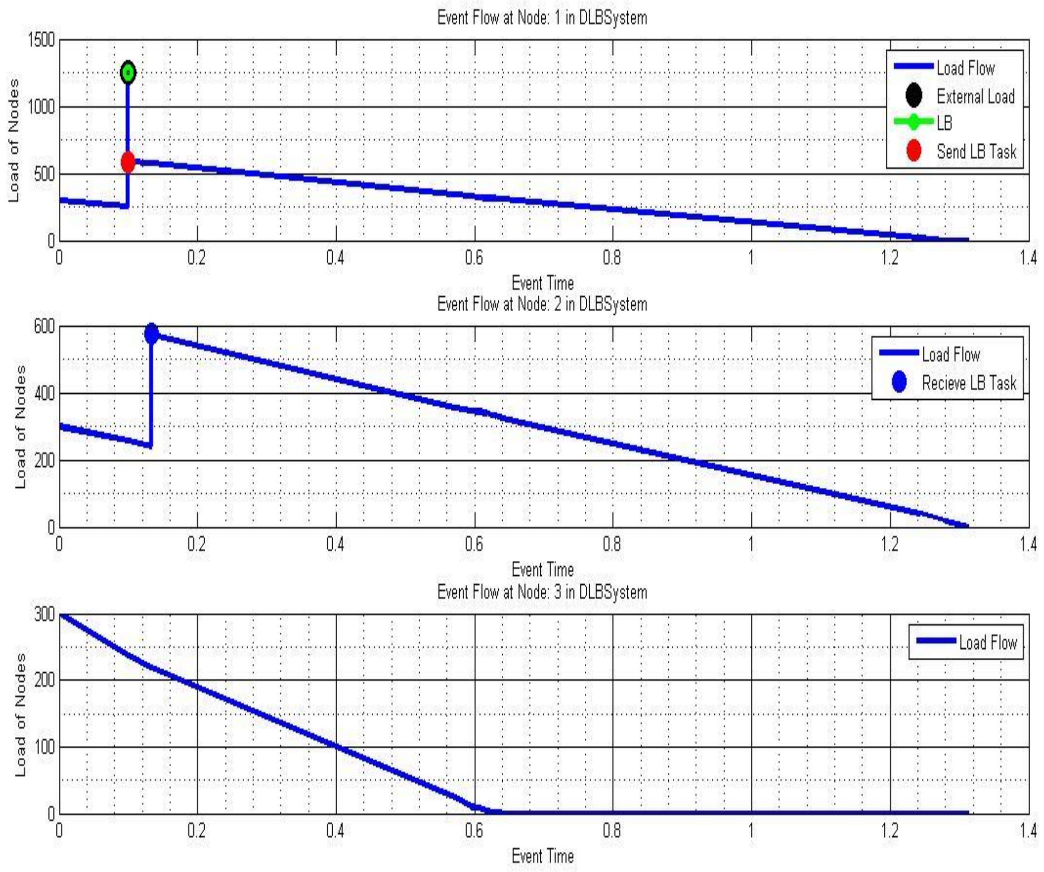


Figure 5.2: the task execution process versus time in network when load balancing is not performed

It should be noted that the instantaneous load in each node is depicted so a linear decay instead of random or Poisson distribution is observed. As seen in figure 5.2, the external load is inserted to node 1 at time t=0.1. Although due to lack of load balancing, this node does not transfer its external load to the neighbor nodes (node 2 and 5 according to considered topology).

In second methodology, we perform load balancing but we do not consider the load-transfer delay according to what is presented in [47]. The execution process of tasks in the network is as follows:



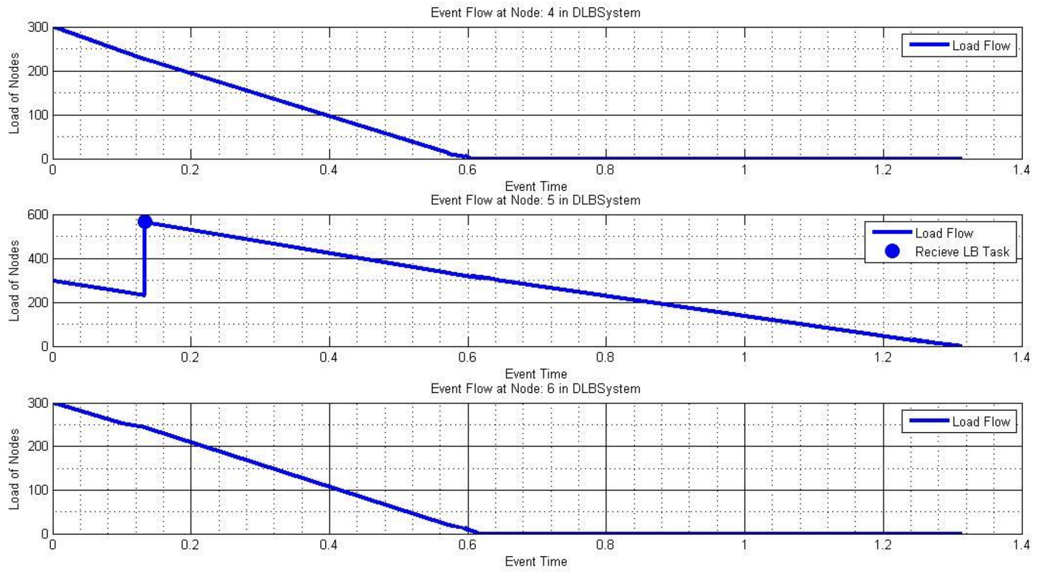


Figure 5.3: the task execution process versus time in the network when load balancing is performed without load transfer delay considerations

In this simulation each node performs the load balancing without considerations of load transfer delay. According to simulated network topology, node 2 and 5 are the neighbors of node 1. Therefore, the external load is transferred to node 2 and 5. Due to load balancing the execution time of node 1, 2 and 5 is almost the same. If our network was fully connected (all nodes where directly connected to each other), the execution time of all nodes after load balancing were the same. As depicted in figure 5.3, it is obvious that the total execution time is reduced after load balancing.

As mentioned, in performed simulation above, we did not consider the load-transfer delay from node 1 to node 2 and node 5 that is 0.033 in load balancing. Even considering this delay in load balancing, the results would be the same. Because the neighbor nodes (node 2 and 5) had initial load and these recourses were fully used till the arrival of external load.

To illustrate the effect of load transfer delay, we consider another scenario where all nodes have 300 initial tasks. The load execution speed is considered to be 200 tasks per second for each node and 100 tasks per second are transferred from one node to another. In this scenario, we assume that the external load is inserted to node 6 having node 5 as its sole neighbor. The results are depicted in figure 5.4.

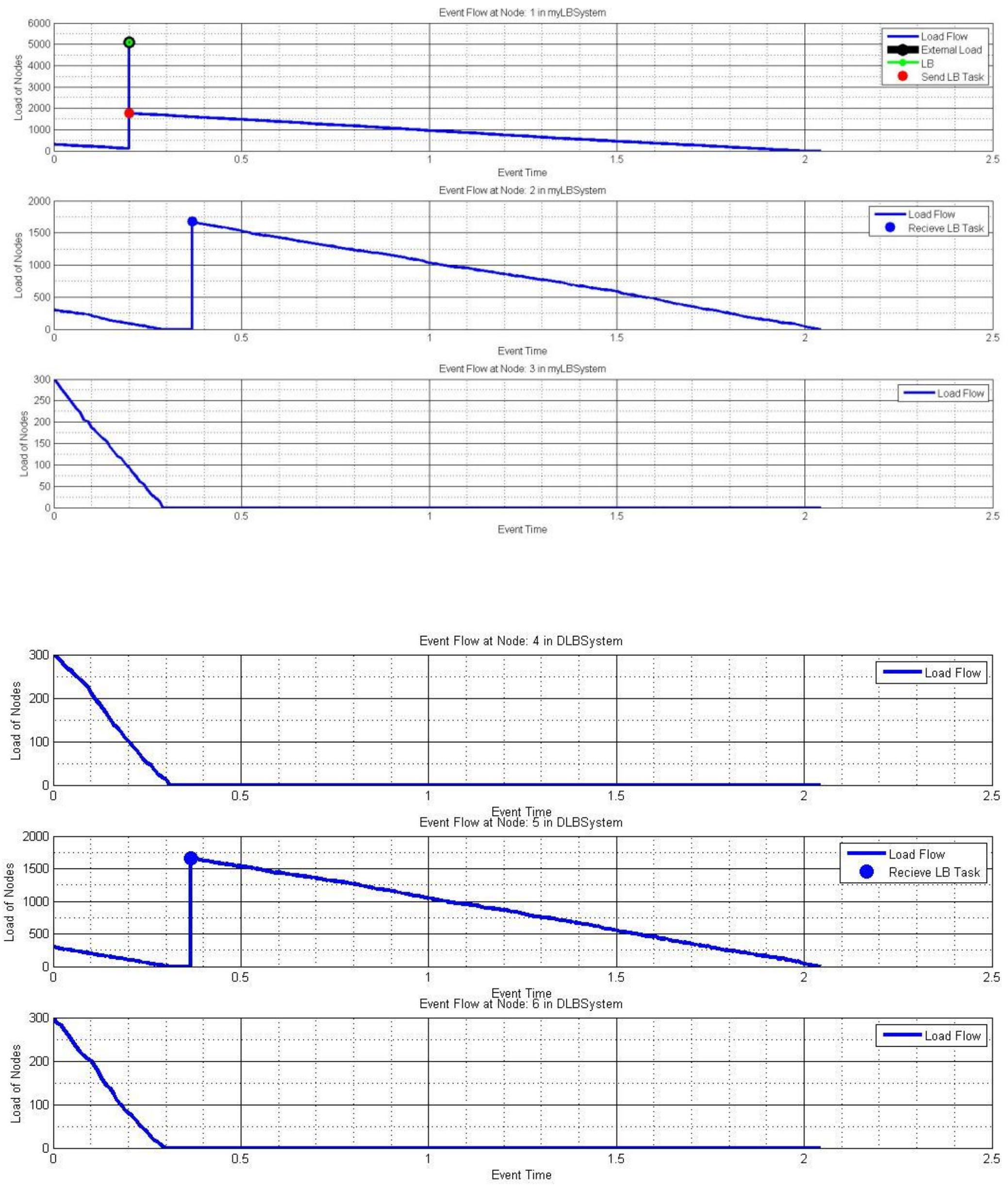
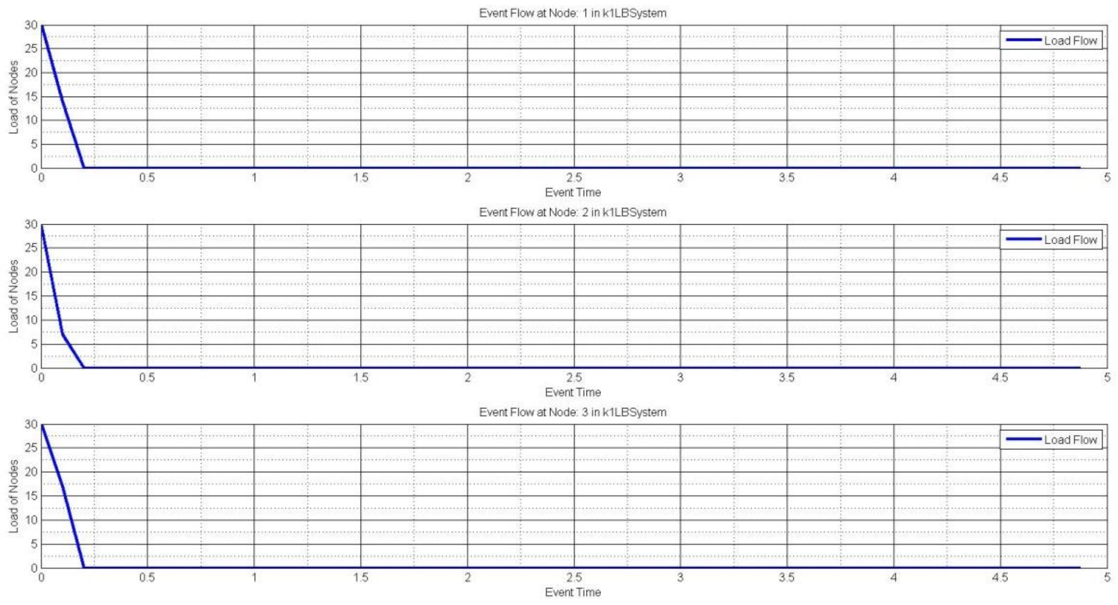


Figure 5.4: the effect of load transfer delay on idle time of systems.

As depicted in figure 5.4, the node 4 performs all assigned tasks at t=0.17 and is in standby state till t=5.05 when the load from load balancing is reached to node 4. The load balancing is performed in a way that all nodes finish their tasks together. The standby time of node 4 causes extra delay in the system and node 4 finish its assigned tasks at t=7.13 while node 6 ends the tasks at t=2.24. According to the definition of total exertion time, the network execution time for assigned tasks is equal to t=7.13. So, it is obvious to compensate for this load transfer delay and to prevent the standby time in the system.

Now, we evaluate the performance of this network with our proposed methods. Here, we evaluate the performance of our first proposed algorithm in reducing the total execution time. The simulation results are as follows in figure 5.5.



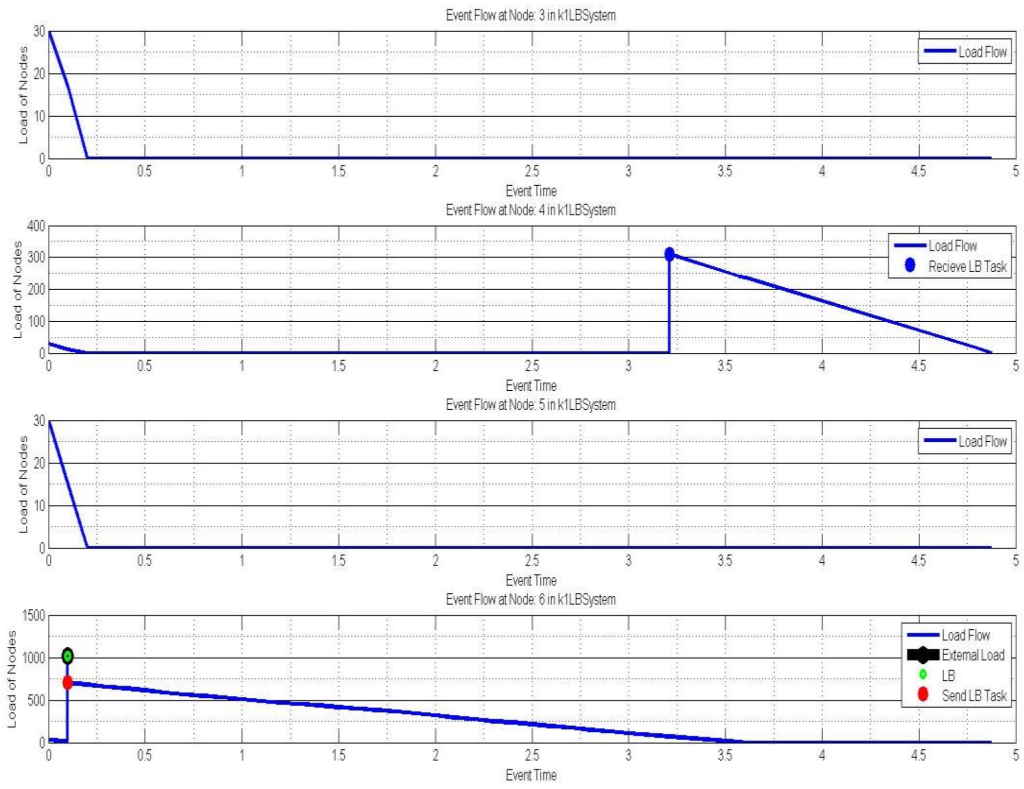


Figure 5.5: The performance evaluation of our first proposed method on total task execution time

As depicted in Figure 5.6, considering the load transfer delay and applying our first proposed algorithm, node 6 transfers 311 tasks to node 4 instead of previous 495 tasks. This causes a significant reduction of standby time for node 4 (3 second instead of previous 5 second). So the total task execution time of system reduces to 5 seconds.

Because of same task execution speed in source and destination nodes, applying the second and third proposed algorithms, results the almost same time.

When the source and destination nodes have different task execution speed, our first proposed algorithm is not the optimum solution. For instance, considering the same network topology with a slight change in improving the source node execution speed from 200 to 280 tasks per second, using the compensator is evaluated and illustrated as following:

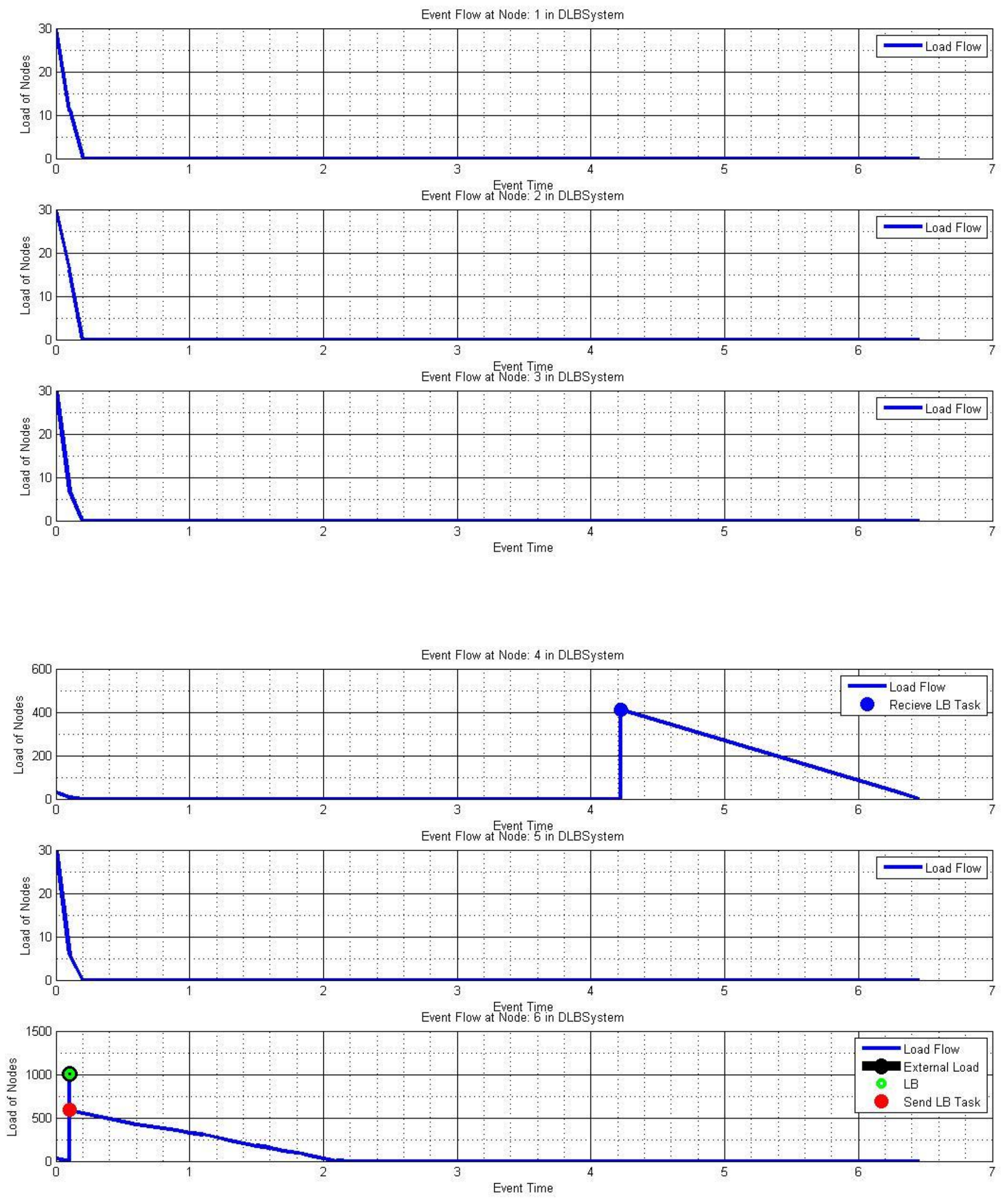


Figure 5.6: the illustration of load execution time where source and destination nodes have different processing speed and no compensation is used.

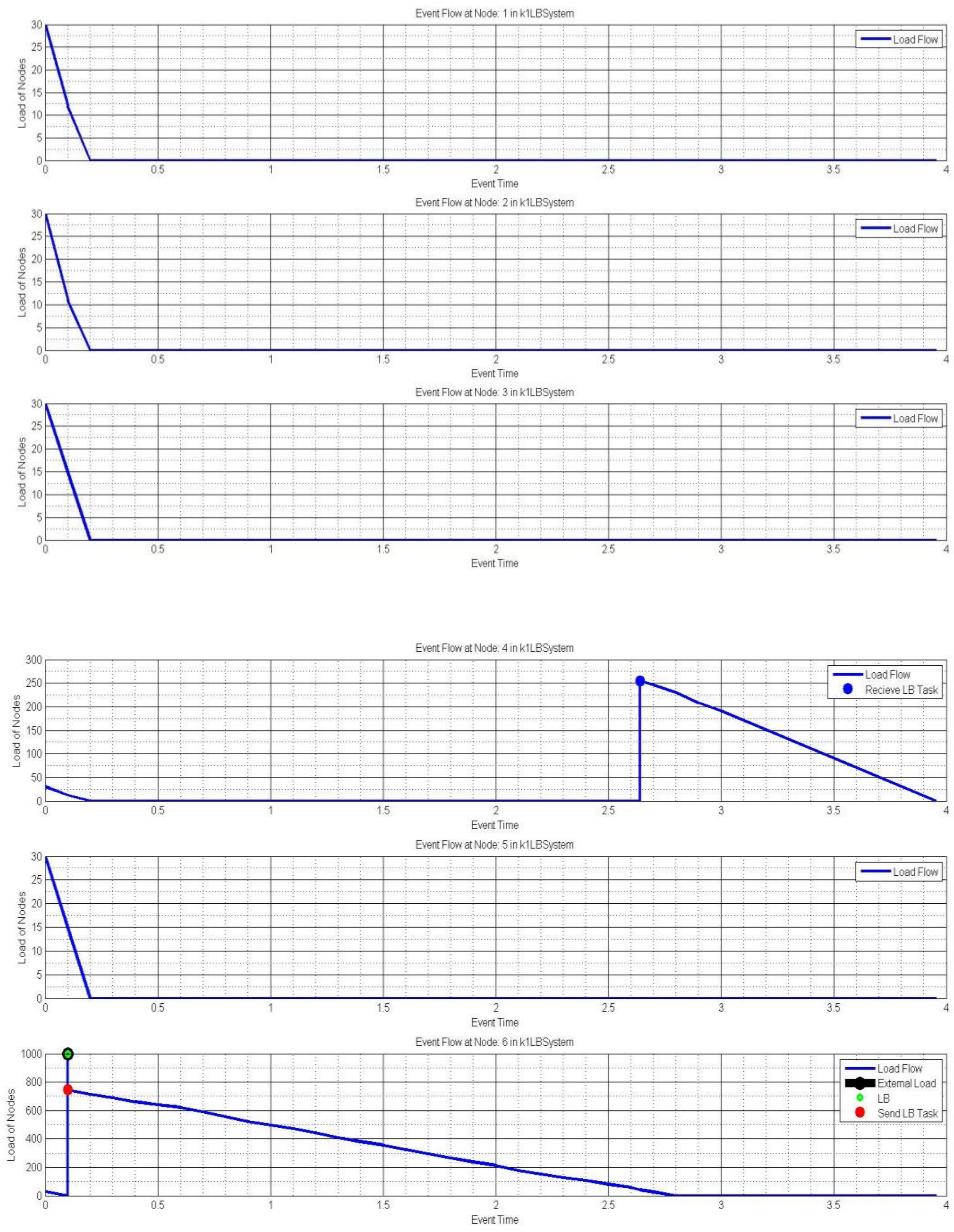


Figure 5.7: performance evaluation of first proposed method on total load execution time where source and destination nodes have different processing speed and no compensation is used

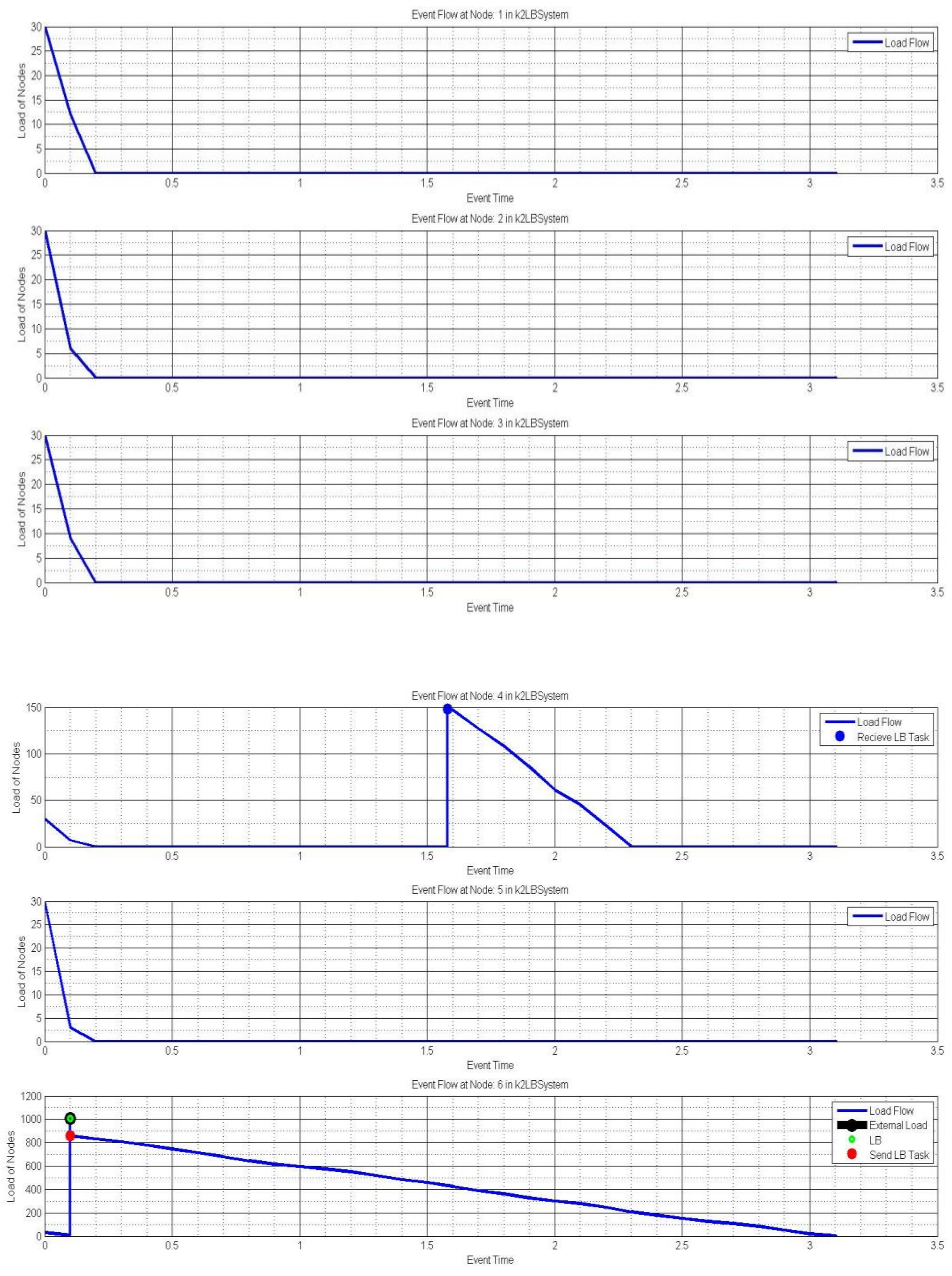


Figure 5.8: performance evaluation of second proposed method on total load execution time where source and destination nodes have different processing speed and no compensation is used

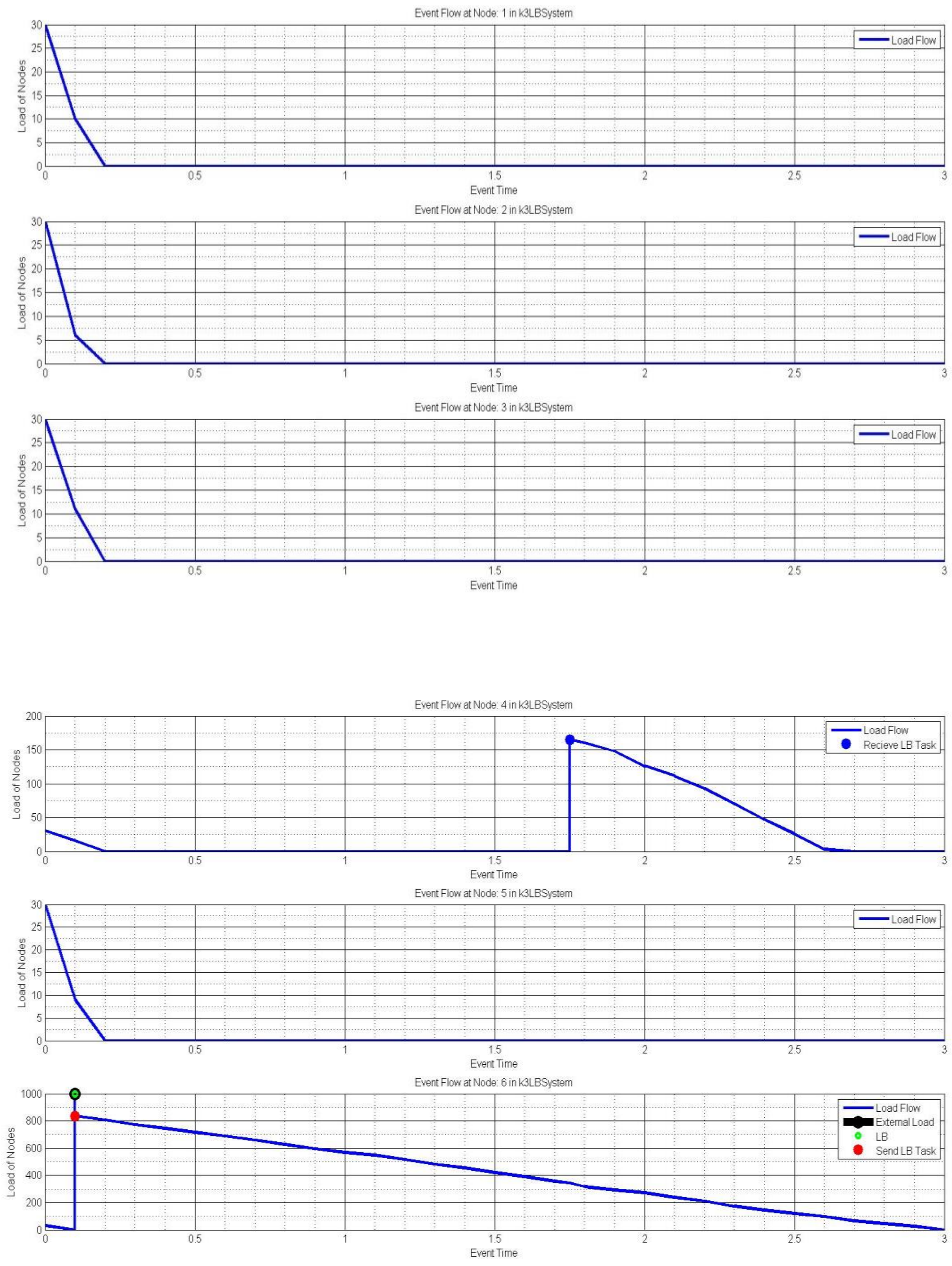


Figure 5.9: performance evaluation of third proposed method on total load execution time where source and destination nodes have different processing speed and no compensation is used.

Considering the above figures, it is obvious that when source and destination nodes have different processing speeds for task execution, our proposed second and third algorithm perform better compared to our proposed first algorithm and all previous works.

To more illustration of compensation effect and our proposed algorithm, we perform the Monte Carlo simulation for late considered network topology. We choose the compensator value k between 0 and 1 and repeat each simulation for 10 times. The total execution time for tasks using this Monte Carlo simulation is depicted as the following:

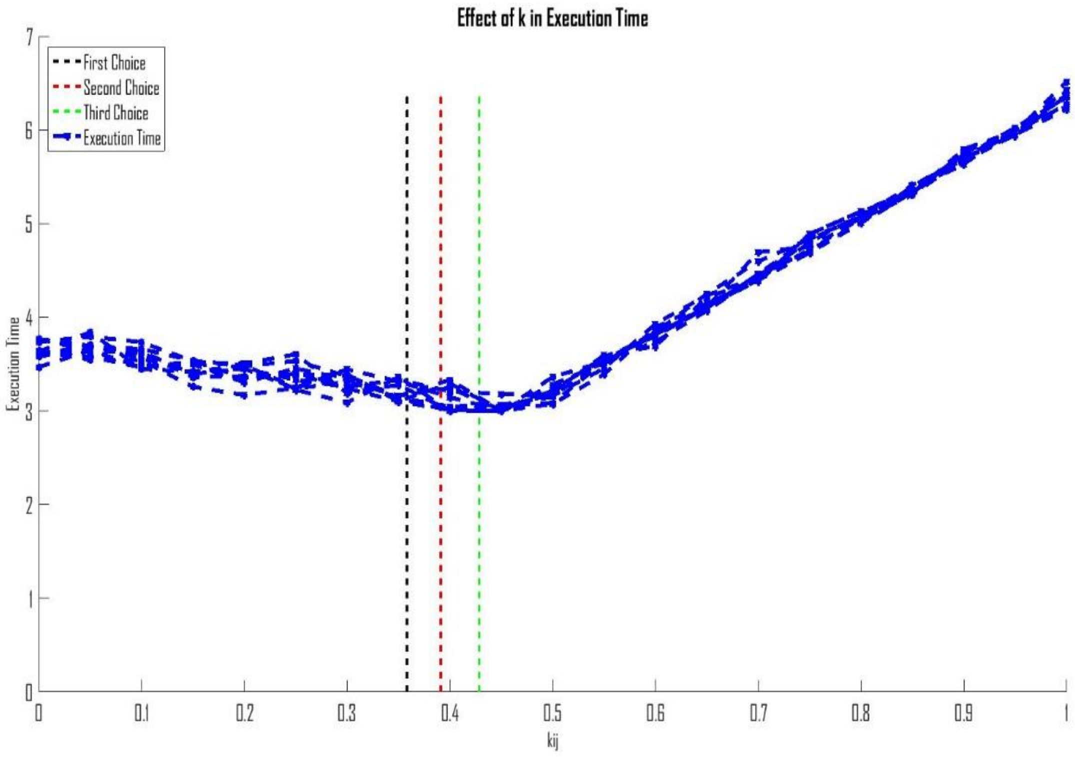


Figure 5.10: effect of compensator value in the total execution time

As depicted in figure 5.10 our proposed third algorithm performs best among our proposed algorithms due to the consideration of number of unexecuted tasks.

To illustrate the effect load execution speed and load transfer speed between nodes, we use the same topology as our previous evaluation in the load execution time. In this part only load execution time is considered and the standby time is neglected.

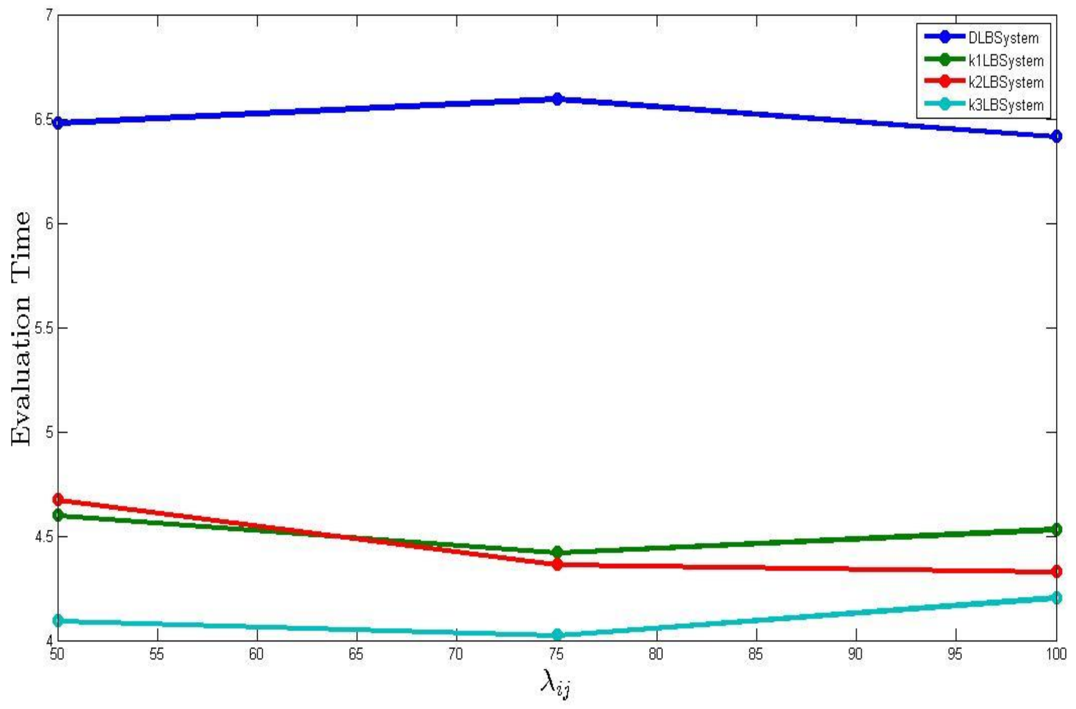


Figure 5.11: effect of different load transfer speed on the load execution time.

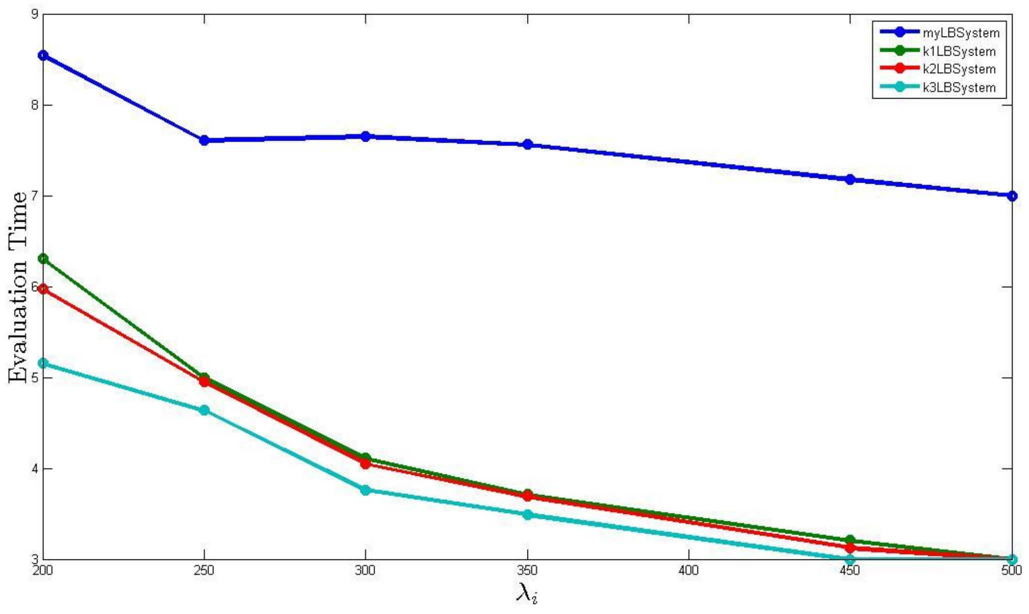


Figure 5.12: effect of different load execution speed on the load execution time

As depicted in figure 5.11 and figure 5.12, increasing the load execution speed in nodes, the load execution time decreases. Although in all cases our proposed methods performs better in compared to the case where load transfer delay is not considered in load balancing.

We now investigate the performance of our proposed method by considering the more complex scenario while our underlying platform be heterogeneous and the execution speed up and also transformation delay and numerous number of load entrances. For this reason, first of all we consider a network with 10 nodes that its topology is changed randomly and also we simulated a random mechanism for entering 1000 external load to the system in different unpredictable time. Command execution speed up is modeled by exponential distribution with average of λ and 0.1λ variance. The transfer of load speed up between nodes is modeled by exponential distribution with average of $\hat{\lambda}$ و and $0.1\hat{\lambda}$ variance. The figure 5.13

show the performance of suggested methods for different ratio of $\hat{\lambda}/\lambda$. For smaller ratio of $\hat{\lambda}/\lambda$, transmission delay times between nodes will be less important, then the proposed methods have the similar performance and functionality of the mentioned method by Dhakal and etc.[47].

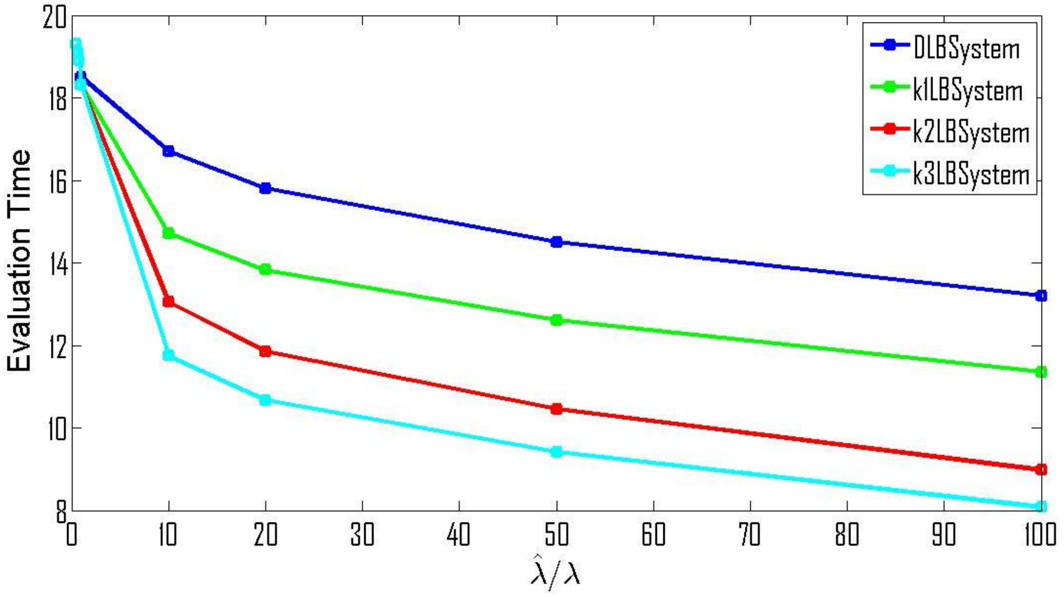


Figure 5.13: The effect of execution speed up ration and load transferring speed up in execution of the load.

For considering impact of number of nodes in performance of proposed method, we define a new scenario. In this scenario the average command execution speed up and the transfer speed up are model based on the Gaussian distribution, like previous scenario. The average execution speed up is initialized by 500 and average transfer speed up is initialized by 50 and also the number of external loads to suit the size of the network is considered as much as 100 times the number of network nodes. The figure 5.13 show the result of this scenario based on different size of network. As can be seen, the proposed method because of its distributed nature of the network size will not impact its proper performance. As can be seen, the proposed method because of its distributed nature of the network size will not impact its proper performance. As can be seen, in the proposed method because of its distributed nature, the network size do not impact its proper performance.

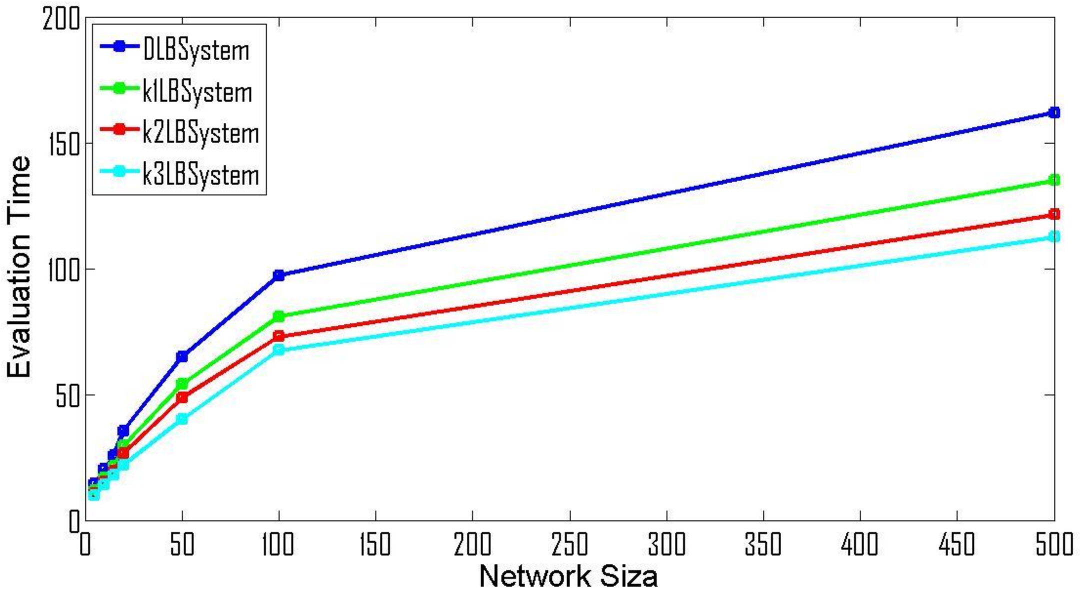


Figure 5.14: The impact of network size in performance of proposed methods.

5 Conclusion

In this thesis a model is proposed that each node can calculate the number of available tasks at the specific time period and each node can estimate its status based on this model and is placed in the two sets, the nodes with positive load and the nodes with negative load. The nodes with negative load, relative to their computing power and volume of load that is intended to them to run, have more capacity to execute the tasks and it's possible to assign more tasks to them. On the other hand, the nodes with positive load has less computing power than the considered load for them and a portion of their load should be transferred to other nodes. Practically this load balancing mechanism can be introduced for estimating amount of transferred load from node with positive load to node with negative load.

We present a model for accurate estimating of extra load in each node that should be transferred and this model is designed to work distributed and there is not ant central node to manage the load of the system and in this model we consider the communication delay in specifying the amount of load should be transferred. For considering the communication delay factor, we define a parameter named Compensating factor and formulate it and then we proposed three methods to determine it while the performance of the proposed load balancing mechanism increase, especially in the systems that the delay of communication is noticeable. The result of simulation illustrates the substantial improvement in the efficiency of the proposed load balancing mechanism in compare with the common distributed load balancing mechanisms in managing dynamic and unpredictable requests.

6 References

1. O. Feinerman, A. Korman, “Theoretical Distributed Computing Meets Biology: A Review”, Distributed Computing and Internet Technology, Lecture Notes in Computer Science Volume 7753, pp 1-18, 2013 .
2. J. Chen, A.H. Sayed, “On the Learning Behavior of Adaptive Networks-Part I: Transient Analysis”, arXiv preprint arXiv:1312.7581, 2014 .
3. J. Chen, A.H Sayed, “On the Learning Behavior of Adaptive Networks-Part II: Performance Analysis”, arXiv preprint arXiv:1312.7580, 2014 .
4. J. Xu, Y. Song, M. van der Schaar, “Sharing in Networks of Strategic Agents”, IEEE J. Sel. Topics Signal Process. - Special issue on "Signal Processing for Social Networks”, 2014 .
5. J. Dongarra, P. Beckman, et al, “International Exascale Software Project Roadmap”. UT-CS-10-652, 2011 .
6. G. Mani, S. Y. Berkovich, L. A. Mihai, “Combinatorial Distributed Architecture for Exascale Computing”, International Conference on Advanced Computing, 2012 Fourth International Conference on Advanced Computing (ICoAC), Chennai, 2012 .
    A. S. Tanenbaum, M. V. Steen, “Distributed Systems: Principles and Paradigms”, Prentice Hall; US edition, January 15, 2002 .
7. R. Baert, C. Lanch, P. Coene, M. D'Hondt, Z. Ma, R. Wuyts, “The Future is Dynamic-Adaptive Runtime Resource Management For Heterogeneous Computer Platforms”, 24th ACM SIGPLAN conference companion on Object Oriented Programming Systems Languages and Applications - OOPSLA pages:733-734, Orlando, Florida, USA, 25-29 October 2009 .
8. P. Kogge, K. Bergman, et.al, “ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems”, Technical report 2008-13, 2008 .
9. J. A. Åström, A. Carter, J. Hetherington, K. Ioakimidis, E. Lindahl, G. Mozdzynski, R. W. Nash, P. Schlatter, A. Signell, J. Westerholm, “Preparing Scientific Application Software for Exascale Computing, Applied Parallel and Scientific Computing”, Lecture Notes in Computer Science, vol. 7782, pp 27-42, 2013.
10. S. L. Mirtaheri, E. Mousavi Khaneghah, L. Grandinetti, M. Shrifi, “A Mathematical Model for Empowerment of Beowulf Clusters for Exascale Computing”, International Conference on High Performance Computing & Simulation, July 01 – 05, 2013, Helsinki, Finland.

11. X. Qin, H. Jiang, A. Manzanares, X. Ruan, S. Yin, “Dynamic load balancing for I/O-intensive applications on clusters”, IEEE Transaction on Storage, vol. 5, no. 3, 2009 .
12. Abdallah, Chaouki T., Ghanem, J., Dhakal, S., Hayat, M. M., Jerez, H.“Load balancing in distributed systems with large time delays: Theory experiment,” IEEE/CSS 12th Mediterranean Conference on Control and Automation, 2004.
13. N. Mohamed and J. Al-Jaroodi, “Delay-tolerant dynamic load balancing,” in Proceedings of the 13th IEEE International Conference on High
14. Performance Computing and Communications (HPCC '11), pp. 237–245, IEEE, Banff, Canada, September 2011 .
15. S. Ashby, P. Beckman, “The Opportunities and Challenges of Exascale Computing report”, US department of Energy, 2010 .
16. C. Li, M. Hurfin, Y. Wang, Reaching Approximate Byzantine Consensus In Partially-Connected Mobile Networks, Research Report 7984, Research Centre Rennes – Bretagne Atlantique, 2012.
17. B. S. Deepak, S. V. Shashikala, K. R. Radhika, “Load Balancing Techniques in Cloud Computing: A Study”, International Journal of Computer Applications, 0975 – 8887, 2014 .
18. C. Xu, F. C. Lau, “Load Balancing in Parallel Computers: Theory and Practice”, Kluwer Academic Publishers, Norwell, MA, USA, 1997 .
19. M. F. Ali, “The Study On Load Balancing Strategies In Distributed Computing System”, International Journal of Computer Science & Engineering Survey (IJCSES) 04/2012; Vol.3,(No.2):19-30, 2012
20. G. Bell, T. Sterling, “Beowulf Cluster Computing with Linux”, The MIT Press, 1 edition, October 1, 2001.
21. J. A. Reading, D. Morris, “Load balancing system and method”, US Patent 7657590 B2, 2010 .
22. D. M. Abdelkader, F. Omara, “Dynamic task scheduling algorithm with load balancing for heterogeneous computing system”, Egyptian Informatics Journal, Vol. 13, no. 2, July 2012 .
23. S. Agarwala, C. Poellabauer, J. Kong, K. Schwan, M. Wolf, “System-Level Resource Monitoring in High-Performance Computing Environments”, Journal of Grid Computing, Vol. 1, no. 3, pp 273-289, 2003.
24. S. L. Mirtaheri, E. Mousavi Khaneghah, A. S. Memaripour, L. Grandinetti, Mohsen Sharifi, Zarrintaj Bornaee, “MULTICS and Plan 9, the Big Bangs in Distributed Computing Systems’ Universe”, IEEE Computing in Science and Engineering (ISI), ISSN :1521-9615, DOI:10.1109/MCSE.2014.3, 2014.
25. K. Wang, K. Brandstatter, I. Raicu, SimMatrix: SIMulator for MAny-Task computing execution fabRIc at eXascale. In Proceedings of the High

Performance Computing Symposium (HPC '13). Society for Computer Simulation International, San Diego, CA, USA, 2013 .

26. R. Wuyts, K. Meerbergen, P. Costanza, " Reactive rebalancing for scientific simulations running on exascale high performance computers", International Conference on Parallel Computing, Belgium, 2012 .
27. S. Jha, "Next Generation Middleware for Distributed Exascale Computing Infrastructure: The Role of Modeling and Simulation", Workshop on Modeling & Simulation of Exascale Systems & Applications, USA, 2013.
28. S. Eyerman, L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads", IEEE Micro, Vol. 28, no. 3, 2008.
29. S. Abu Elenin, M. Kitakami, "Performance Analysis of Static Load Balancing in Grid", International Journal of Electrical & Computer Sciences, Vol. 11, no. 03, 2011 .
30. J. Dongarra, P. Beckman, et al., "The International Exascale Software Roadmap", International Journal of High Performance Computer Applications, Vol. 25, no. 1, 2011 .
31. X. Qin , H. Jiang , A. Manzanares , X. Ruan , S. Yin , "Dynamic Load Balancing for I/O-Intensive Applications on Clusters", Journal ACM Transactions on Storage, Vol. 5, no. 3, 2009.
32. S. Tannenbaum, "Distributed Operating Systems", Prentice Hall, US Ed., 2005 .
33. S. Tanenbaum, M. V. Steen, "Distributed Systems: Principles and Paradigms", Pearson; 2 edition, 2006.
34. M. Paksoy, J. Prado, "Comparing centralized and decentralized distributed execution systems," Internet: http://www.sccs.swarthmore.edu/users/07/mustpaks/distsys_paper.pdf, 2015.
35. S. Rajarajeswari, Dr. M. Aramudhan, "Centralized Load Balancing Mechanism in Federated Cloud", Vol. V, no. VIII, 2015 .
36. M. Nandagopal, V. Rhymend Uthariaraj, "Decentralized Dynamic Load Balancing for Multi Cluster Grid Environment", Advanced Computing, Vol. 133, pp 149-160, 2011.
37. S. Yun, A. Proutiere, "Distributed Load Balancing in Heterogeneous Systems", 48th Annual Conference on Information Sciences and Systems, CISS 2014, 2014 .
38. N. R. Chopde, "Load Balancing Dynamically in Distributed and Parallel Computing", International Journal of Current Engineering and Technology, Vol.5, no.1, 2015.
39. Darji, J. Shah, R. Mehta, "Survey paper on various load balancing algorithms in cloud computing", International Journal of Scientific & Engineering Research, Vol. 5, no. 5, 2014.

40. M. J. Zaki, S. Parthasarathy, W. Li, “Customized Dynamic Load Balancing for a Network of Workstations”, University of Rochester Technical 602, 1995.
41. S. M. Lau, Q. Lu, K.-S. Leung, “Adaptive Load Distribution Algorithms for Heterogeneous Distributed Systems with Multiple Task Classes”, Journal of Parallel and Distributed Computing, Vol. 66, no. 2, pp. 163-180, 2006.
42. K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, L. G. Gervasio, “New challenges in dynamic load balancing”, Applied Numerical Mathematics - APPL NUMER MATH , vol. 52, no. 2-3, pp. 133-152, 2005 .
43. R. Yahaya, M. Latip, A. Othman, A. Abdullah, “Dynamic Load Balancing Policy with Communication and Computation Elements in Grid Computing with Multi-Agent System”, International Journal on New Computer Architectures and Their Applications (IJNCAA), Vol. 1, no. 3, 2011.
44. A. Grosu, T. Chronopoulos, “Noncooperative load balancing in distributed systems”,Journal of Parallel and Distributed Computing, Vol. 65, no. 9, 2005.
45. Y. Murata, T. Inaba, H. Takizawa, H. Kobayashi, “A distributed and cooperative load balancing mechanism for large-scale P2P systems”, International Symposium on Applications and the Internet Workshops, Phoenix, USA, 2006 .
46. K. Wang, K. Brandstatter, I. Raicu, “SimMatrix: SIMulator for MAny-Task computing execution fabRIc at eXascale”, the High Performance Computing Symposium (HPC '13), Society for Computer Simulation International, San Diego, CA, USA, 2013.
47. H. Menon, L. Kalé, “A distributed dynamic load balancer for iterative applications”, International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13). ACM, New York, NY, USA, 2013 .
48. T. L. Casavant, J. G. Kuhl. “A taxonomy of scheduling in general-purpose distributed computing systems”, IEEE Transaction on Software Engineering, Vol. 14, no. 2, 1988 .
49. S. Dhakal, M. M. Hayat, J. E. Pezoa, Y. Cundong, D. A. Bader, “Dynamic Load Balancing in Distributed Systems in the Presence of Delays: A Regeneration-Theory Approach”, IEEE Transactions on Parallel and Distributed Systems,, vol. 18, pp. 485-497, 2007.