In this chapter,

2.1 Introduction
2.2 Load Balancing Classification
   2.2.1 Location Based Classification
   2.2.2 Destination Based Classification
   2.2.3 Initiation Based Classification
   2.2.4 Content Based Classification
   2.2.5 Response Path Based Classification
2.3 Heterogeneity in Distributed Computing Environment
2.4 Content Aware Scheduling in Distributed Computing Environment
2.5 Use of Open Source Software in Distributed System
2.6 Dynamic Load Balancing in Routers
2.7 Dynamic Load Balancing in Multi-Core Processors
2.8 Summary

2.1 INTRODUCTION

The use of Dynamic Load Balancing (DLB) in distributed computing environment is an important research issue. Researchers have shown keen interest in developing optimal and efficient DLB techniques for these distributed computing environments. Whether it is a cluster, grid, cloud or network of workstations, DLB is an important tool for improving the overall system performance.

During past two decades, DLB has attracted the researchers as a variety of perspective. With the rise of parallel and distributed computing, DLB
has become more important for several applications viz. task scheduling, scientific computing, partial differential equation, structural analysis of molecular dynamics, sparse matrix computation, finite element applications, graph theory, data mining, parallel discreet simulation, computational fluid dynamics, computational machines and more [Couny, 1994; Hendricson, 2000]. Complex security issues and commercial applications also drew their attention in the area of load balancing and scheduling. The researchers concern about the DLB can be noticed by observing the trend of past few years where number of papers related to DLB has been published in the Journals and conference proceedings. Moreover several workshops, special conferences and journal issues are dedicated to dynamic load balancing and scheduling in distributed computing environment.

Three load balancing policies are almost common in all the proposed algorithms. These are Information Policy, Distribution Policy and Rebalance policy. The architecture of generic load balancer has been shown in Figure 2.1. Using information policy, load balancer collects the load and other information regarding the system, distribution policy forwards the client requests to the servers based on the information collected by the information policy. The rebalance policy, balances the load on the servers and nodes using process migration.

Our survey is limited itself to the scheduling and load balancing in clusters, routers and multi-core processors. The successive sections, highlight the studies made in the past decades in these areas and will also present our view towards the ongoing research on use of dynamic load balancing in the area of distributed computing environment.

2.2 LOAD BALANCING CLASSIFICATION

Researchers have classified load balancing in several ways. This section covers the classifications based on the following criteria:
• Location Based Classification
• Destination Based Classification
• Initiation Based Classification
• Content Based Classification
• Response Path Based Classification

2.2.1 Location Based Classification

This classification is based on the location where the load balancing decision is made and can be categorized as: central policy and distributed policy. In central policy a central node is named as load balancer where all the scheduling decisions are made. In distributed policy, all the participating nodes act as load balancer. Whereas centralized policy is simple and effective, its failure may stop working of whole system. Decentralized policy is a complex fault tolerance load balancing policy having great potential to exploit heterogeneity and scalability. Studies show that the centralized policy may give better result in homogeneous system whereas decentralized policy is suitable for heterogeneous environment [Paksoy, 2006; Razzaque, 2007].
Similar location based strategies found in the literature include Global and Local load balancing strategies. In global strategy, load balancer keeps the load status of all participating nodes whereas in local strategy, participating nodes are partitioned into small groups and load information is exchanged within the group only. The performance of the system using global strategy or local strategy depends on the nature of application and dynamic behavior of nodes. Global strategy generates large number of messages between load balancer and nodes in the system resulting increased communication overhead. Local strategy minimizes communication overheads but suffers performance degradation due to lack of synchronization among the groups. In a system using local load balancing strategy, it is possible that under loaded nodes of one group are unavailable to the over loaded nodes of the other group [Xu, 2007].

Cardellini et al. have classified web server architecture into local and global scale out approach where the difference lies in the geographical locations of the nodes in the cluster. In global approach the nodes of the cluster are located at different geographical locations while in local load balancing approach the nodes are present in the same location [Cardellini, 2002].

2.2.2 Destination Based Classification

The classification falls under the category of how the load balancing is achieved in the situation where node allocation is performed by schedulers and load balancers. In the literature, these allocation policies are referred as placement policy, destination policy or location policy. Alakeel has proposed Random, Probing and Negotiation placement policies. In random policy, a node is selected randomly as destination node by the scheduler and accepts the client’s request if its index value is less than threshold value, otherwise a different node is selected by the scheduler in the similar way. In probing, all the participating nodes are partitioned to form groups. A node of a randomly selected group is identified as destination node using polling technique i.e. a node having better response time compared to other nodes of the group. Negotiation placement policy is the widely
used by the DLB algorithms. In this policy, a negotiation takes place among all the nodes to become the destination node. Bidding and Drafting are two well known negotiation policies [Alakeel, 2010].

A distributed engine with a fuzzy rule based load balancing model for service process is presented by Cao et al. They employed a heuristic algorithm to evaluate and indicate the potential load of a client’s request with a load tag. Later, based on the load tag of an incoming client’s request and load statuses of the engine, the controller of the load balancer allocates the request to an appropriate service according to a set of fuzzy rules [Cao, 2008]. The heuristic methods are distinguished by different policies for process migration and threshold update where artificial intelligence can be effectively used in implementation of DLB. Algorithms have also been suggested to deal with deadline scheduling and job priorities [Fu, 2003].

Load balancers also use quantitative measures of the load known as load indices. The common indices used by the researchers are queue length, memory utilization, processor utilization etc. The success of a load balancer depends on how accurately it predicts the value of these indices. However, the resource multiplicity and heterogeneity makes it difficult to predict the load. Kartza treated a node as a queuing network where arrivals and resource consumptions follow probabilistic patterns. Mean inter arrival time and execution times of processes are based on exponential distribution. He suggested that in a system of M/M/1 server model, the probabilities of busyness can be attached with the nodes of cluster and predicting the probability is relatively easy [Kartza, 2003]. Load balancers are needed in the systems where load on the system varies in unpredictable ways i.e. the main objective of the load balancers is to stabilize such systems by distributing the work evenly. Success of a load balancer depends on managing the service measures like mean response ratio, overhead cost, waiting queue management etc.
Process migration is one of the critical criteria used by the load balancers. Modeling and theoretical studies of DCS have suggested that performance of the system can be enhanced by using process migration. Preemptive and non-preemptive process migration has been compared and justifications have been given that the performance of the systems using non-preemptive scheduling is better than the one using preemptive. The similar result is found in both homogeneous as well as heterogeneous DCS [Jiang, 2004]. In some situations simple algorithms give better performance in terms of average response time as compare to complex algorithms. Eager et al. have shown that a server initiated algorithm may perform better, if the system is under heavy load and cost of migration of a process is not much higher than that of transfer of an arriving job [Eager, 1986].

Senbel presented a dynamic self-adapting system for a server cluster that supports multiple services with different priorities. The number of servers in each service changes dynamically in response to load fluctuations. In proposed system, each node runs three mechanisms: the request routing mechanism, the neighborhood table maintenance mechanism and the membership control mechanism. The request routing mechanism handles incoming requests and decides whether it can process the request or re-direct it to a neighbor. The neighborhood table maintenance mechanism periodically updates its neighbor table by dissipating its information to a certain number of other nodes. The membership control mechanism decides whether each node remains in the service group, switches to another service, or joins the idle group. Senbel’s main contribution is a new membership control technique based upon maintaining a local leader for each group [Senbel, 2010].

Harchol-Balter et al. have presented task migration policy for DCS based on process migration where they have compared preemptive and non-preemptive process migration strategies using process lifetime distribution for DLB. They found that performance of the system is far better than non-
preemptive [Harchol-Balter, 1997; Harchol-Balter, 1999; Harchol-Balter, 2003]. Well known processes address transfer policies covered in the literature are:

- **Eager-Copy**: state and address space of the migration process is transferred before process execution starts on the destination node.
- **Lazy-Copy**: transfers only the necessary information to resume process execution on the destination node.
- **Pre-Copy**: Migrating process is not suspended on the source node until most of its address space is transferred to the destination node.
- **Post-Copy**: transfers minimum number of pages from source node to destination node and destination node starts the execution only after receiving rest of the address spaces.
- **Flushing**: A file server is used to support virtual memory as backup and dirty pages are flushed to the file server after suspension of the processes on the source node.

### 2.2.3 Initiation Based Classification

The classification is based on who is taking the decision of load balancing. Cardellini et al. has proposed: client-based scheduling, Domain Name System (DNS)-based scheduling, dispatcher-based scheduling and server-based scheduling; where destination server is identified by the corresponding scheduling node i.e. either client or DNS or scheduler or server itself [Cardellini, 1999].

Zeng et al. proposed a load balancing strategy for the object storage architecture. In their work, they introduced general application server architecture and co-related it with object storage architecture. The characteristics of storage object in object storage system enabled them to apply their load balancing strategy at the back end of servers or applications in order to deal with bottlenecks at the storage tier in data-intensive applications [Zeng, 2005].
2.2.4 Content Based Classification

A similar classification is suggested on content-awareness and content-blindness load balancing based on the OSI layer, where load balancing is performed. In content-blind policy, load balancing is managed at data link layer or network layer whereas in content-aware load balancing policy, application layer is involved. Choi proposed a classification based on the level where load balancing is applied hardware level, system software level, middleware software level and application software level [Choi, 2004].

Static load balancing is the simplest form of scheduling where all the required information for load balancer is known in advance and remains unchanged till the execution of algorithm. Dynamic load balancing is static load balancing over time and load balancer does not have complete knowledge of the set of inputs and works on theory of probability. Generally, these inputs are received in future and are not necessarily related to the inputs previously received. For each time slot, some parameters of the system may change. However, these changes are expected by the algorithm. For example, load on various servers of a cluster at any moment may change in next time slot. The dynamic load balancing algorithm must consider these changed parameters for performing the optimal scheduling over time.

With the limited number of IPv4 addresses and the limitation of protocol, Friedrich et al. have shown how applications can be migrated to next generation Internet protocol IPv6 for nodes of cluster as well as load balancers. They have used Loaded, a user-space server load balancer for IPv4 and IPv6 networks and demonstrated that the performance loss due to additional user/kernel mode switches is acceptable [Friedrich, 2006].

Most of the scheduling algorithms ignore delay based strategies and the jobs are dispatched immediately by the scheduler even if the node is fully overloaded. In this situation, if scheduler receives a large number of requests, the existing as well as newly arrived requests suffer from poor response time. Hui and
Chanson suggested that scheduler should delay the newly arrived processes until at least one of the currently running processes terminates on the heavily loaded node. The delay strategy is generic and can be incorporated in all kind of scheduling algorithms [Hui, 1997]. For heavily loaded websites, cluster architecture is one of the popular solutions.

2.2.5 Response Path Based Classification

An alternate classification of cluster load balancing is based on the return path of the client response: Two-way architecture and one-way architecture. In two-way architecture, response is forwarded to the client through the load balancer by the node of the cluster whereas in one-way architecture, response is forwarded to the client directly by the processing node of the cluster rather than passing through load balancer. Combining constant degree topology's characteristic, XiaoHai et al. introduced an efficient load balancing algorithm based on Reverse Spanning Tree (REST) for heterogeneous nodes. Without additional data structure, REST uses local routing path information where a new node can identify the number of virtual servers it should run on, and then join all virtual servers on the spanning tree peer set [XiaoHai, 2010].

Aversa et al. presented a prototype of scalable web server that consists of cluster of nodes that collectively accept and service TCP connections through a load-balancer. The host IP addresses are advertised using the Round Robin DNS (RR-DNS) technique, allowing hosts to receive requests from clients. Once a client attempts to establish a TCP connection with one of the hosts, a decision is made to redirect the connection to the node of the cluster with the least number of established connections. In this prototype, they used low-overhead Distributed Packet Rewriting (DPR) technique to redirect TCP connections. The load information is maintained using periodic multicast amongst the cluster hosts [Aversa, 2000].
2.3 Heterogeneity in Distributed Computing Environment

The laboratory work and real time simulation results indicate the importance of load balancing for improving the performance of distributed systems. Scientists have different opinion about various issues in DCE like index selection policy, migration policy and load rebalance policy etc. Owing to the variation in assumptions, platforms, architectures and benchmarks [Tresco, 2005]. Number of strategies and algorithms has been proposed by the research community for the issues arising due to heterogeneity in distributed computing environment.

Effective work load distribution depends on the nature of the underlying system i.e. homogeneous system or heterogeneous system. System heterogeneity may exist in the system due to various factors such as CPU speed, main memory size, disk capacity, FSB speed etc. A load balancing algorithm dealing with heterogeneity should incorporate impact of each factor. While considering heterogeneous clusters, Guist et al. described that different computing powers of the participating nodes can be manipulated for initial workload distribution. Static load balancing can be used where execution time and other characteristics of the request is known in advance. In other cases where expected values of these characteristics are being used, only dynamic load balancing algorithms can be used [Guist, 2005].

Rebalancing in heterogeneous DCE is one of the important activities where load of the heterogeneous nodes is redistributed through process migration. The activity may be performed on periodic basis, push basis or pull basis. In periodic policy, load balancer collects the load index information of nodes at regular time interval and reallocates the processes. This policy is effective for lightly loaded systems. If a node is not responding, load balancer removes it from the system. Similarly, this technique is useful for the systems where nodes are joining and leaving the system as and when required. Push and pull policies are event driven policies. In push policy which is a sender initiated policy, a heavily loaded node sends request to the load balancer, to migrate some of the processes.
to relatively lightly loaded nodes. In pull policy which is a receiver initiated policy, an ideal or under loaded node of the system sends the messages to the other nodes of the system. As message contains current load of requesting node, the nodes of the system compare their load with the requesting system and migrates the processes accordingly. It has been stated in the literature that sender initiated algorithm behaves well when the system lightly or moderately loaded. For heavily loaded systems, receiver initiated algorithm performs well [Aweya, 2002; Alakeel, 2010]. Authors also proposed an adaptive strategy which switches its behavior from push to pull and vice versa based on the situation.

A common approach used for load balancing on cluster of nodes is to predict future performance based on the past performance. Zhong and Coworkers have used processor queue length for load estimation of heterogeneous nodes and found a marginal improvement in the throughput over static load balancing [Zhong, 1997]. Eager et al. and Waldspruger suggested single threshold approach for process transfer and various policies of DLB in distributed computing [Eager, 1986; Waldspruger, 1992]. Double threshold load transfer policy was proposed by Alonso and Cova [Alonso, 1988]. Load sharing ensures that no node is idle or underutilized while other nodes are over loaded. Task allocation policy, on the other hand, considers different modules of a single task on different processing elements.

Kontogiannis et al. presented an adaptive load balancing algorithm for cluster-based web systems that uses web server’s load as well as network condition. They have introduced load balancing policy based on two criteria; HTTP process time and network delay. HTTP process time describes web servers ability to process a forthcoming request, while network delay estimates network conditions. They compared their algorithm with known stateless and stateful algorithms. They used state blind balancing algorithms: Round Robin (RR), Weighted Round Robin (WRR) and state full algorithms such as Least Connections (LC). Their research distinguishes between network congestion and load instances [Kontogiannis, 2009].
Li et al. used queuing theory as the strategy for load balancing in server cluster and proposed a queuing theory based self-adaptive algorithm. The algorithm collects the load of each server, calculates the expected response time of client’s request and provides the information to the load balancer for cluster management. They have shown the efficiency of the proposed algorithm through comparison with dynamic self-adaptive algorithms in the homo-structure and hetero-structure environment [Li, 2008]. Cheung presented a dynamic fuzzy load balancing system running in a distributed object computing environment based on Jini. He collected system parameters in a fuzzy sense and established a set of fuzzy inference rules based on these parameters. Cheung has shown how fuzzy logic can be effectively used as a tool for load balancing [Cheung, 2001].

From the above literature review, it is observed that most of the DLB algorithms available for heterogeneous environment have considered processor speed of the participating nodes. Only few algorithms have taken into account the other heterogeneity factors but due to their complexity, the algorithms are infeasible. The DLB algorithms proposed in this research incorporate most of the heterogeneity factors highly sensitive to individual nodes.

2.4 Content Aware Scheduling in Distributed Computing Environment

Content aware load balancing uses the information stored in the client request as the criteria for selecting a particular node of cluster. In workload aware LB, files on the nodes of cluster are categorized as core and part. The files accessed most frequently are termed as core files and rest of the files are referred as part files. In workload aware load balancing, the request for the core files can be responded by any node of the cluster but request for part files can be responded by few designated nodes [Cherkasova, 2001]. The frequency based portioning of files and their locality in the nodes of the cluster improves the throughput of the system. Pai et al. have introduced Locality Aware Request Distribution (LARD) strategy for clusters. In LARD, high locality is achieved through the main memory cache.
of the back ends by dynamically dividing the backend nodes for specific tasks to improve the cache hit rate [Pai, 1998]. In LARD, a specific task is served by a specific node of a cluster which is the bottleneck or single point failure of the system. To avoid this problem, Pai et al. have proposed LARD with Replication (LARD/R), where the nodes to respond the specific requests are replicated.

Casalicchio and Colajanni have proposed Client Aware Policy (CAP) where client requests are classified on the basis of their expected impact on components of the nodes of cluster. This classification partitions websites located on the nodes of the cluster as: Web-Publishing, Web-Transaction, Web-Commerce and Web-Multimedia.

- Web-Publishing: websites consist of webpage with simple database access.
- Web-Transaction: websites consist of webpage with complex database access.
- Web-Commerce: websites where information is provided to the clients using secure protocols like SSL and HTTPS.
- Web-Multimedia: Websites serving audio and video requests.

Authors compared the performance CAP with RR and LARD and shown that CAP gives better cache hit rate for both static and dynamic Websites [Casalicchio, 2001].

Lin et al. have proposed three content aware strategies: Content Aware Weighted Least Load (CAWLL), Extended Locality Aware Request Distribution with Replication Policy (xLARD/R) and Content Aware Hybrid Request Distribution (CAHRD). In CAWLL, the value of workload of requests is determined by resource consumption and least loaded backend node is selected. The weight is assigned to each node of the cluster and this weight value is set as the average number of response packets sent for a particular type of request. Lin et al. extended LARD/R strategy to include the cost of connection handoff. The connection handoff is the process of transferring the connection information among the client, frontend node and backend node. The xLARD/R schedule the
client requests based on the total load of frontend node and backend node. CAHRD combines the strategies offered by CWLL and xLARD/R [Lin, 2009].

Chiang et al. have modified WARD and proposed two content aware policies: Content-Based Workload-Aware Request Distribution with core Replication (CWARD/R) and Content-Based Workload-Aware Request Distribution with Frequency based Replication (CWARD/FR). Similar to WARD/R, CWARD/R classifies the files of the node into core and part files. Core files are replicated to each end node of the cluster while the part files are kept with the designated nodes and their utility. The URI of each request is examined by the scheduler and determines whether the request is for core or part file and accordingly schedules the request. The core theme of CWARD/FR is similar to CWARD/R except more frequently accessed files are replicated in most of the replicated servers and rest of the files are kept in few servers. The performance of system using CWARD/R and CWARD/FR are more or less similar [Chiang, 2008]. In another study, Chiang et al. have modified client aware policy and locality aware request distribution policy and proposed Group Client Aware Policy (GCAP) and Locality Aware Request Distribution with Replication and Classification (LARD/RC). The GCAP uses the similar classification as the one used by CAP and differs as different types of request are scheduled to a specific server according to server's capability using weighted round robin. LARD/RC also uses the CAP classification and designates each server to serve only specific types of request based on server's capability [Chiang, 2009].

Literature review presented in this section, focuses on content aware load balancing to schedule the requests and used only well known scheduling algorithms to forward the client requests to the serves. To the best of our knowledge, none of the researchers have considered scalability, heterogeneity and the way server responds to the clients all together. Chapter 3 proposes a content aware DLB algorithm which takes care of heterogeneity of web server cluster along with scalability and one way architecture to respond client’s request. Chapter 4 proposes a content aware DLB algorithm which implemented and two
stage load balancing strategy and provided reserved service rates to the client requests. In first stage, algorithm uses linear programming approach for optimal use of resources of the WSC and in the second stage push or pull rebalancing strategies are being used.

2.5 USE OF OPEN SOURCE SOFTWARE IN DISTRIBUTED SYSTEM

Wenzheng et al. proposed a load balancing algorithms using every working server, to send corresponding ask signals to load balancer. The ask signal is sent on the basis of Discrete Uniform Distribution (DUD) principle and then balancer assigned user requests accordingly to ask queue. In their research, they have shown that the proposed algorithm is easy to implement, fast and guarantees the even distribution of the workload [Wenzheng, 2010]. Gill et al. presented TAO, a next-generation open source load balancer that supports server-side transparency, state migration, fault tolerance, run-time control over replica of servers by using the corresponding services of CORBA [Gill, 2002].

Many task-based programming models have been developed using open source software and refined to support application development for shared memory platforms. Asynchronous tasks are powerful programming abstraction that offers flexibility in conjunction with great express. Research involving standardized programming models like OpenMP and non-standardized models like Cilk facilitate improvements in many programming implementations. A number of approaches to implementing tasking constructs have been reported, but fewer attentions to providing the user with capabilities for fine tuning with the execution of their code. LaGrone et al. have presented an overview of OpenMP implementation, highlighted its main features and discussed its implementation with user controlled runtime variables [LaGrone, 2011]. Chapter 5 presents a generic adoptive load balancing algorithm which can easily be incorporated in existing and new open source software.
Table 2.1 Organization of Cluster Load Balancing

<table>
<thead>
<tr>
<th></th>
<th>Content Blind</th>
<th>Content Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One Way</strong></td>
<td>- Data Link Layer Forwarding (Direct Routing)</td>
<td>- TCP Connection Hop</td>
</tr>
<tr>
<td></td>
<td>- Network Layer Forwarding (IP Tunneling)</td>
<td>- Socket Cloning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TCP Rebuilding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Multiple TCP Rebuilding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Multiple Connection TCP Hand-off</td>
</tr>
<tr>
<td><strong>Two Way</strong></td>
<td>- Network Layer Forwarding using NAT</td>
<td>- TCP Connection Binding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TCP Splicing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Redirect Flows</td>
</tr>
<tr>
<td><strong>DLB Algorithms</strong></td>
<td>- Random</td>
<td>LARD, extLARD, CAP,</td>
</tr>
<tr>
<td></td>
<td>- Round Robin (RR)</td>
<td>WARD, xLARD/R, CAHRD,</td>
</tr>
<tr>
<td></td>
<td>- Weighted Round Robin (WRR)</td>
<td>DEQAL</td>
</tr>
<tr>
<td></td>
<td>- Least Connections (LC)</td>
<td>CWARD/CR, CWARD/FR,</td>
</tr>
<tr>
<td></td>
<td>- Weighted Least Connections (WLC)</td>
<td>EQUILOAD, ADAPTLOAD,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CAWLL, IQRD etc.</td>
</tr>
</tbody>
</table>

2.6 DYNAMIC LOAD BALANCING IN ROUTERS

Singh et al. introduced Globally Oblivious Adaptive Locally (GOAL), a load-balanced adaptive routing algorithm for a network having high path diversity, offering many alternative paths. GOAL has achieved global load balancing by randomly choosing the direction to route in various dimension and local load balancing is then achieved adaptively by routing the selected directions [Singh, 2003]. A scalable and decentralized content aware dispatching algorithm is proposed by Xu. et al. The algorithm uses distributed hash based dispatching mechanism to solve scalability problem of centralized algorithms and reduced communication overhead of distributed algorithms [Xu, 2007].

The common design goals of Routing Algorithms are correctness, optimality, simplicity and low overhead, stability and robustness, rapid convergence and flexibility. QoS routing covers routing information and routing algorithm. Routing algorithms can be classified based on different criteria:
information source, number of destinations, decision place, number of considered QoS metrics, required QoS guarantee levels, wired or wireless (fixed or mobile).

Transier et al. have proposed a dynamic load balancing routing algorithm for mobile ad-hoc networks in addition to greedy position-based algorithms. It uses the current load of the nodes to decide regarding the next forwarding hop along with the current positions of the neighbors. Their algorithm determines the current load by scanning the medium at the MAC layer in regular intervals. In his work, Transier found that the average connection delay is reduced which is beneficial for higher path lengths [Transier, 2005].

It has been shown that multi-path routing balances the load more effectively than single-path routing. Contrary to this, Ganjali and Keshavanian have shown that this is not necessary. They introduced a model of evaluating the load balancer under multi-path routing for some shortest paths and shown that unless a large number of paths participate (which is very costly and therefore infeasible) the load distribution is almost similar to single shortest path routing [Ganjali, 2004]. Dhamdhere and Dovrolis have shown their worry over the use of small buffers in Internet routers. Small buffers reduce queuing delays, while maintaining full utilization at the target link at the cost of a high loss rate. It harms the performance of certain applications and results in lower throughput for most large TCP transfers. They raised an important open issue in the buffer sizing problem, the tradeoff between loss rate and queuing delay in terms of application-layer performance [Dhamdhere, 2006].

In his paper, Yang has proposed Adaptive Routing Control (ARC) algorithm based on Dijkstra's algorithm to find the efficient and flexible path of workflow. He considered two factors: the available bandwidth of a link and the anticipated waiting time of a node to make the workflow system more flexible. Although it is difficult to predict the waiting times of nodes, He computed node's cost by considering the average node processing time and number of waiting tasks in the queue [Yang, 2003].
Computational change inside the network can improve the overall performance of network applications. Pradhan et al provided an efficient system support for placing computation in a network router. They identified a set of requirements related to resource control, scheduling, protection and efficiency which are relevant to the designing of a routing algorithm [Pradhan, 2001].

Karimi et al. have proposed a system consisting of heterogeneous nodes in terms of CPU, memory, disk, switches etc. and involves routing table, load index, cost table and Fuzzy controller for effective load balancing. Routing table is used to identify optimal route, load index determines the load of nodes of the system and Fuzzy controller determines the status of nodes i.e. node is a sender, receiver or neutral using fuzzy inference rules. Cost table provides nodes communication cost and number of heavily loaded nodes [Karimi, 2009].

In their letter, Simsarian et al. have proposed a routing architecture that combines load balancing with optical switching. They demonstrated that load balancing reduces the complexity of the switch fabric and simplifies the scheduling. Furthermore, their proposed architecture allows the router to scale to large throughput efficiently due to non-blocking nature of individual stages [Simsarian, 2004].

Although some research has been carried out on Routing security, still several security threats are not addressed or partially addressed in the routing protocols. A framework has been proposed by Hung for secure link state routing protocol based on information level data origin authentication and information-level confidentiality. The framework comprises of multiple virtual trust routing domains to avoid severe security threats. Hung has proposed a three stage research plan for secure link state routing protocol: collecting security issues, proposing a secure link state framework and investigating the implementation issues [Hung, 2003].

As mentioned in this section, research issues in routing involved link level load balancing, switching level load balancing and various security issues.
Chapter 6 proposes an adoptive DLB algorithm which balances load at input queue by providing virtual input queue and uses linear programming to schedule the incoming client requests.

2.7 Dynamic Load Balancing in Multi-Core Processors

The speed mismatches between processor and disks in cluster of servers have become major bottleneck that deteriorate the performance. Load balancing schemes detects I/O imbalance on nodes of the cluster and take a decision to migrate the I/O requests from overloaded nodes to relatively under loaded nodes. The decisions are based on data availability and remote access overheads. Other than I/O load, the scheme also takes into account processor and memory load, maintaining same level of performance as the existing one [Qin, 2004].

Tan et al. have presented a programming and execution model for multi-core architectures with memory hierarchy. They addressed the challenge of exploiting fine grain parallelism and locality of non serial dynamic programming on a multi-core architecture and proposed a parallel pipelined algorithm for filling the dynamic programming matrix by decomposing the computation operators. The parallel algorithm tolerates the memory access latency using multi-thread and is easily improved with tile technique [Tan, 2007]. In his paper, Patterson explained the consequences of incorporating smaller and more numerous transistors into one giant microprocessor, resulting in a device that couldn't compute faster than the chip it was replacing. The improvement in the processing capability can be achieved by assembling them into multiple microprocessor cores [Patterson, 2010].

Chen et al. have explored the problem faced by the graphics processing units (GPU) and found that the programming techniques employed on these GPUs were not sufficient to address problems exhibiting irregular and unbalanced workload. The problem was exacerbated when trying to effectively exploit multiple GPUs concurrently. They proposed a task-based dynamic load-balancing solution for single and multi-GPU systems [Chen, 2010]. Ishfaq et al.
have addressed the problem of power-aware scheduling of tasks onto heterogeneous and homogeneous multi-core processor architectures with the objective of scheduling to minimize the energy consumption as well as the make span of computationally intensive problems. According to authors, the multi-objective optimization problem can be solved by conventional approaches that try to maximize a single objective [Ishfaq, 2010].

Asymmetric Multiprocessor Scheduler (AMPS) contains three components: asymmetry-aware load balancing that balances threads to the cores in proportionate to their computing power; faster-core-first scheduling that schedules threads to faster cores whenever they are under-utilized; and NUMA-aware migration, that controls thread migrations based on predictions of their overheads. These components complement one another, collectively providing efficient support for asymmetric architectures. AMPS can be deployed by modifying existing operating systems and no changes in applications [Li, 2007].

As multi-core processors with tens or hundreds of cores begin to proliferate, system optimization issues will become important to all programmers. However, unlike with High Performance Computing (HPC) community, the focus of the multi-core programmer will be on programming productivity and portability as much as performance. Rajagopalan et al. have introduced a novel scheduling framework for multi-core processors that strikes a balance between control over the system and the level of abstraction. Their framework used high-level information supplied by the user to guide thread scheduling and gave the programmers fine control over thread placement [Rajagopalan, 2007].

Literature review shows that load balancing is implemented for symmetric and asymmetric multi-core processors at operating system level. Chapter 7 proposes an application level DLB algorithm which addresses I/O bottleneck problem by using Java NIO and achieves parallelism using Java multithreaded programming.
2.8 Summary

In this Chapter, a comprehensive review of the research in the area of distributed scheduling and resource management for distributed computing environment is presented. The survey reveals that significant work has been done in designing and development of load balancing algorithms for distributed computing elements. This motivates us to pursue the research in the area and address the problems in the areas where further research scope exists. Section 2.2 classified the load balancing strategies found in the literature. Section 2.3 and 2.4 focuses on the heterogeneity factor and content aware load balancing in the literature and Section 2.5 reveals the work done in the area of DLB for open source software. Section 2.6 presents the research in the area of router load balancing and Section 2.7 proposes DLB algorithm for symmetric multi-core processors.