Energy Efficiency through Virtual Machine Redistribution in Telecommunication Infrastructure Nodes

Miraj Hasnaine Tafsir
University of Helsinki
Faculty of Science
Department of Computer Science
Networking and Services
Master´s Thesis
September 2013

HELSINGIN YLIOPISTO − HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta – Fakultet – Faculty
Faculty of Science

Laitos – Institution – Department
Department of Computer Science

Tekijä – Författare – Author
Miraj Hasnaine Tafsir

Työn nimi – Arbetets titel – Title
Energy Efficiency through Virtual Machine Redistribution in Telecommunication Infrastructure Nodes

Oppiaine – Läroämne – Subject
Networking and Services

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
M. Sc. Thesis	22.09.2013	69

Tiivistelmä – Referat – Abstract

Energy efficiency is one of the key factors impacting the green behavior and operational expenses of telecommunication core network operations. This thesis study is aimed for finding out possible technique to reduce energy consumption in telecommunication infrastructure nodes. The study concentrates on traffic management operation (e.g. media stream control, ATM adaptation) within network processors [LeJ03], categorized as control plane.

The control plane of the telecommunication infrastructure node is a custom built high performance cluster which consists of multiple GPPs (General Purpose Processor) interconnected by high-speed and low-latency network. Due to application configurations in particular GPP unit and redundancy issues, energy usage is not optimal.

In this thesis, our approach is to gain elastic capacity within the control plane cluster to reduce power consumption. This scales down and wakes up certain GPP units depending on traffic load situations. For elasticity, our study moves toward the virtual machine (VM) migration technique in the control plane cluster through system virtualization. The traffic load situation triggers VM migration on demand. Virtual machine live migration brings the benefit of enhanced performance and resiliency of the control plane cluster. We compare the state-of-the-art power aware computing resource scheduling in cluster-based nodes with VM migration technique. Our research does not propose any change in data plane architecture as we are mainly concentrating on the control plane. This study shows, VM migration can be an efficient approach to significantly reduce energy consumption in control plane of cluster-based telecommunication infrastructure nodes without interrupting performance/throughput, while guaranteeing full connectivity and maximum link utilization.

ACM Computing Classification System (CCS):
C.3 [Process Control Systems]

Avainsanat – Nyckelord – Keywords
GPP, Control plane, Control layer, Media plane, virtual machine monitor

Säilytyspaikka – Förvaringställe – Where deposited
Kumpula Science Library C-

Muita tietoja – Övriga uppgifter – Additional information

Table of contents

List of figures

1 Introduction

In the age of technology blessed world global warming is an alarming problem. High energy consumption [Bos10] due to telecom and ICT activities is one of the significant reasons for global warming. Rapid growth of the telecommunication area is aimed to provide high processing capacity and throughput in server operations. This enhanced performance will effectively increase the energy usage by infrastructure equipment to perform core network operations (e.g. traffic handling). A number of axes (e.g. energy usage ratio, energy efficiency features and cooling requirement) need to be taken under extensive research effort to achieve the green behavior from telecommunication sector. Here the green behavior refers to minimizing energy consumption through use of energy efficient technologies in telecom activities [Kha12]. Following sections give the driving force behind the study, concrete research issue and methodology followed.

1.1 Motivation of the study

While approaching to minimize overall energy consumption it is better to look back why energy optimization is a burning issue. As for example, very few Facebook users are concerned about the amount of energy (i.e. electricity) being consumed. In fact, every Facebook operation is consuming some amount of electricity in the data center. Increasing amount of electricity consumption in turns results in increasing amount of CO2 emissions, which causes long-term environmental impact [KSK+12]. Here Facebook represents just an example of ICT applications. The similar scenario is also visible in the telecommunication sector. The end user never knows how the services are provided, or handled by core network operations. In wireless networking technologies, the telecom operators always show potential to provide enhanced services along with new application features to their subscribers. Emerging network performance needs enhanced traffic handling in telecommunication core networks. Intensive processing of various traffic-types (e.g. voice, video) entails extra traffic handling capacity and high power consumption by underlying network equipment. For any kind of traffic handling operation, network equipment essentially uses electricity and thus generates CO2 emissions somewhere down the line, consequently causing harm to climate [KSK+12]. Moreover, excessive operation performed by traffic handling components produce huge amount of heat. Larger cooling systems with high cooling capacity are needed to chill the system environment. These cooling systems are also high energy grabber, which

result in the same curse to the climate. Consumable energy by network equipment may be capped at some point of time in future, whereas performance demand always has upward trend. Therefore reducing energy usage is critical, as we cannot compromise with performance for power optimization.

1.2 Research Focus

To facilitate the increasing traffic requirements over the wireless network, the telecommunication infrastructure nodes are being enhanced to serve high throughput and performance. So traffic planes (for managing and processing traffic) in telecommunication nodes are continuously being leveraged for handling and processing more incoming traffic. This traffic operation is one of the most significant energy candidates in telecommunication core network operations. Hence, this thesis focuses on power efficient traffic handling within the telecommunication infrastructure node.

Figure 1.1 below shows the concept of telecommunication infrastructure node. For handling the traffic, typical node type contains two interdependent functional blocks noted as control plane and data plane. Traffic management functionalities on the control plane executes on a centralized processor cluster. General purpose processors (GPP) interconnected by a high-speed and low-latency network are forming such a cluster. The data plane is built up of programmable reduced-function application-specific network processors. But processing decisions are always made by control plane functionalities. And this thesis aims to achieve energy efficient behavior from high computing control plane GPP cluster (that performs traffic management operation).

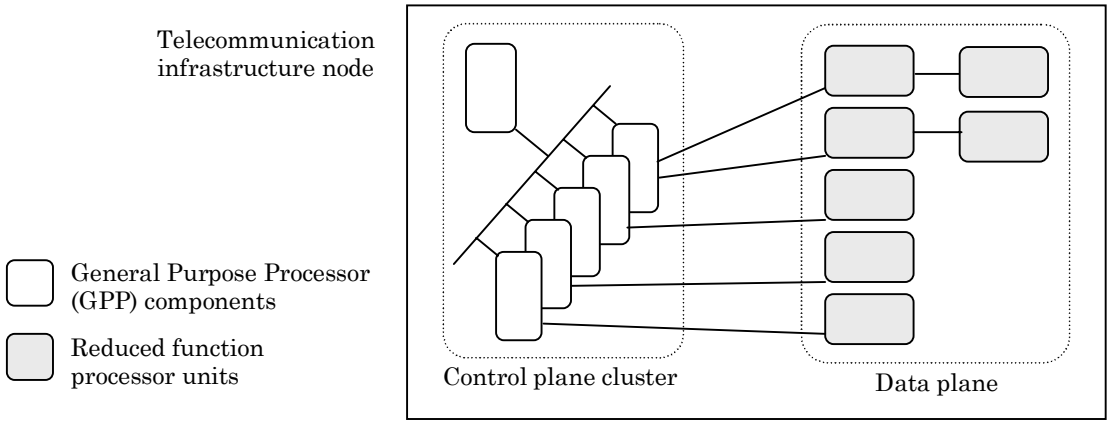

Figure 1.1: Traffic planes structure in cluster-based infrastructure node

Significant confronts can be noted from GPP cluster in control plane. GPPs are configured according to control layer applications. In low traffic situation, shutting

down, or CPU throttling is not possible due to scattered software profiles among GPPs. Additionally there are resiliency issues for handling any crashing situation meaning that, data transmission may be interrupted if any corresponding serving GPP fails. To survive from this kind of situation, control plane architecture provides backup GPP (always remains operational) to support when the primary one fails, consuming energy even in normal operational mode.

1.3 Research Question

The main research question of this thesis is:

- How can energy efficiency be improved especially while controlling user sessions inside telecommunication infrastructure nodes?

This question is narrowed down into another sub question based on research focus area.

- How can energy consumption be reduced in control plane of cluster based telecommunication nodes?

1.4 Study methodology

This thesis is based on the theoretical investigation of several scientific approaches to achieve power efficiency in clusters and data centers. Investigation starts by analyzing the current technology for power-aware computing for resource scheduling in cluster based nodes. We investigate the current control plane (responsible for traffic management) architecture inside the cluster-based telecommunication infrastructure nodes.

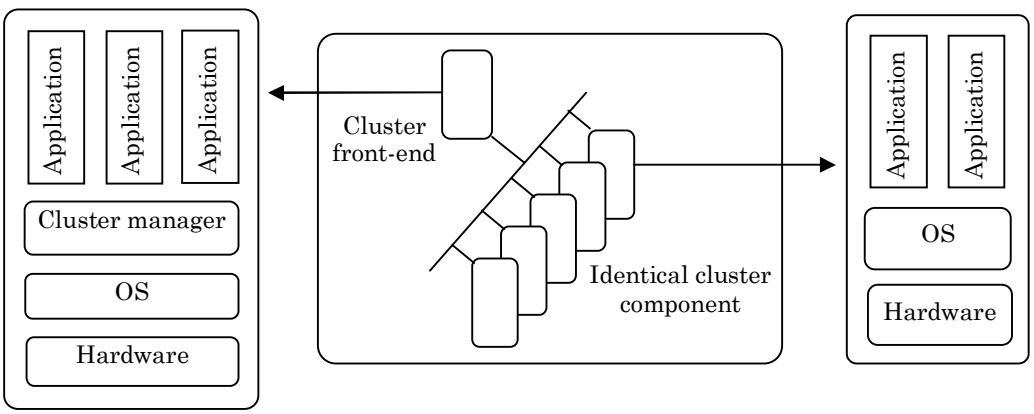

Figure 1.2: Control plane GPP cluster

Figure 1.2 above shows the architectural view of the control plane cluster. As a base

line of this thesis we also analyze the load based resource scheduling technique inside the infrastructure nodes. Methodology of this thesis is divided into two significant sequential study steps to achieve energy efficiency in traffic operations. Firstly we study the virtualization of general purpose processor. Secondly the study continues with the discussion of energy optimization through virtual machine migration. These are introduced in the following two sections.

Virtualization of general purpose processor

In traditional system environment, deploying multiple operating systems on a single set of hardware resources was not trivial. This is because of having difficulty in proper resource sharing among the operating systems as they would require dedicated hardware resources. Using a virtual machine actually resolves this problem, because this technique does not allow the system and applications running in it to directly interact with underlying hardware resources. The virtual machine is essentially a software package that formulates an abstraction of the underlying hardware. Implementation of VM can provide a complete system platform to support execution of an entire operating system with all its applications. Virtual machine interacts with lower level hardware resources through virtual machine monitor (VMM). Virtual machine monitor makes abstraction of functionalities and OS from underlying hardware component through virtual machine. It is a control program that is accountable for proficiently congregating the virtual machines. If we consider privilege level of VMM execution, it is higher than that of operating system supervisor. The Figure 1.3 below depicts the hypothetical concept of GPP virtualization.

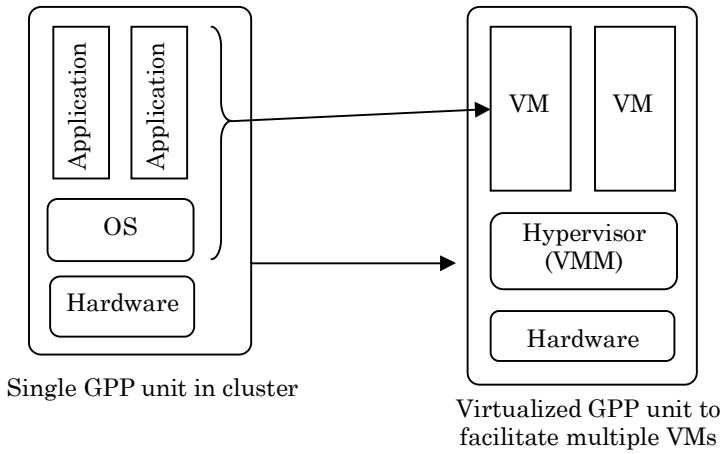

Figure 1.3: Virtualization of GPP unit

Virtual machine monitor is also referred as the Hypervisor. This hypervisor/VMM creates an abstraction layer over the physical hardware to form the virtualized

environment in a particular physical general purpose processor cluster component. Applications (running inside the VMs) never see the interactions with physical hardware resources. The study proceeds with virtual machine based GPP cluster environment in the control plane. We went through several scientific articles about virtualization, hypervisor and live virtual machine migration to analyze the virtualization approach to be applied in the control plane GPP cluster.

Energy optimization through VM migration

Our study proceeded towards the live virtual machine migration process [JDW+09] [CFH+05], which can be a good approach for power-aware traffic management. Several points of VM migration are studied; such as live VM migration with minimal network overhead (due to migration process) and also less engagement of hypervisor (virtual machine monitor) in migration process. These are crucial because we can never compromise with the system performance (for traffic handling) due to energy efficient behavior. Also resiliency situation is one more significant confront to overcome. In our proposed study, the number of virtual machines is kept equal to the number of GPP components in the cluster. As the research goes forward, we see the possible apt way of VM migration without any noticeable functional interruption. According to our hypothetical manner, the working methodology is conventional during the high-traffic situation as every virtual machine located on its original physical component. During low-traffic particular virtual machine residing on certain physical GPP component migrates to another and former GPP goes scaled down to reduce power.

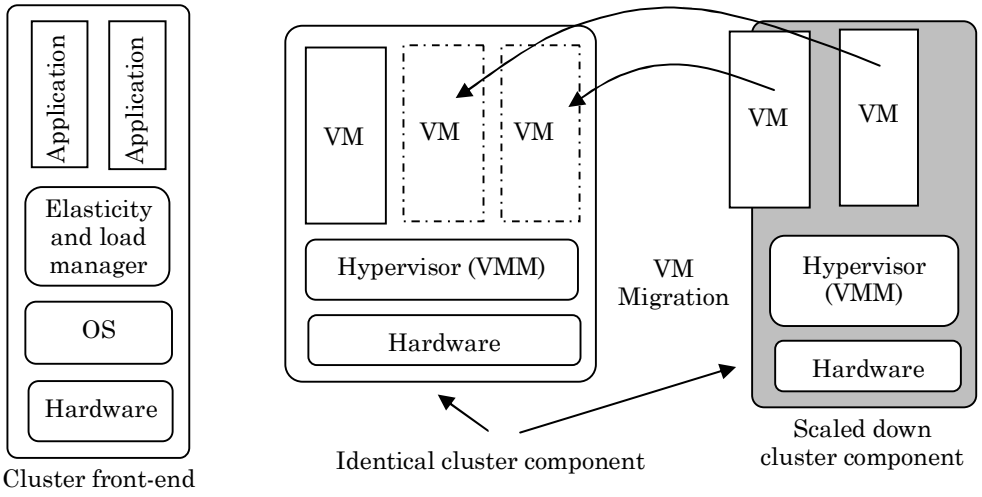

Figure 1.4: Virtual machine migration in control plane cluster

The achievable working tactic after employing virtual machine migration technique (in low traffic situation) is depicted in the Figure 1.4 above. This ensures energy aware

traffic operation in control plane. Migration also depends on the work load situation of the destination component. We theoretically explore possible performance and energy usage of the infrastructure node to provide energy efficient traffic management.

1.5 Layout of the thesis

The layout of the thesis consists of six chapters. The thesis begins with the theoretical discussions of mobile core network which is a part of the network connectivity layer of horizontal telecommunication network architecture [Wit00]. Then we discuss the ideology of cluster based telecommunication nodes followed by the discussion of virtualization concept with a detail analysis of virtual machine types. Then it continues to study elaborately the virtual machine migration, which is opted to follow as the means of energy reduction.

As the thesis proceeds, after illustrating the theoretical concept, the study moves forward to describe the possibility of using virtual machine migration technique in control plane cluster. A mathematical model is shown followed by an algorithm that can be used to make migration decision while squeezing and releasing the cluster size. Apparently, this can achieve the elastic behavior by respectively switching/scaling down and waking/powering up physical cluster components.

Thereafter some limitations and confronts are mentioned, regarding possible dynamic energy efficient solution. As the thesis narrows down the study scope from vast telecommunication network architecture to core infrastructure node, therefore we show some additional possible research axes of power-aware traffic operations.

Lastly the thesis is concluded by summing up the study approach and discusses the answers to the research questions.

2 Theoretical background

In horizontal layered telecommunication architecture, core network is one of the parts of connectivity layer. In this chapter we will give brief illustration of telecommunication layered network architecture concentrating on connectivity layer. After that discussion will continue with the load based scheduling for gaining energy efficiency inside cluster-based infrastructure nodes. As mentioned in introductory chapter, these nodes have hierarchical structure of components to serve traffic demand in the core network architecture. In later section, we will study possible green factors due to core network functionalities. These analyses are significant for this study to narrow down the study scope for focusing on efficient energy usage through traffic plane virtualization inside telecommunication infrastructure nodes.

2.1 Mobile core network

In telecommunication network, user session control and connectivity are separated into different layers. In traditional architectures, call control and connectivity were bundled in telecommunication architecture. Nodes and functionality are arranged in layers according to their specific areas of use in horizontal layered architecture [DRB03]. This separation actually brings a lot of advantages to the network architecture as the functions provided are now independent from each other from layer to layer. Several standardization initiatives such as Megaco [RFC 3015] in the IETF, Tiphon in ETSI [ETSI TS 101 329-2] and the Multiservices Switching Forum (MSF) adopted this separation principle of modern network. In universal mobile telecommunication system (UMTS) this horizontally layered architecture essentially divides the network into three different layers [Wit00].

- Application layer;
- Network control layer; and
- Common connectivity layer.

The Figure 2.1 below illustrates this layered architecture [BBB07] [CGa04]. The end-user applications reside in the *application layer*. Generally in modern networks, mobile terminals and dedicated application servers are the places in the network where applications are implemented. In case of application servers, they are often complemented with content servers. Content servers further host service related

databases or libraries. Application layer interfaces with network control layer through application programming interfaces (APIs). These open APIs are used by application feature developers to implement new services and applications.

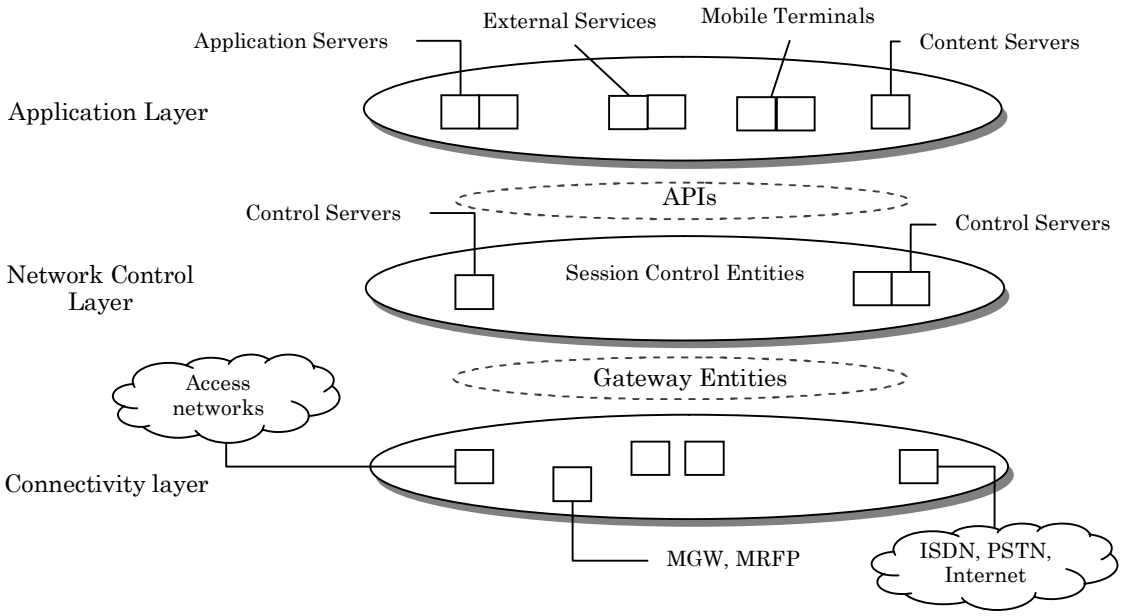

Figure 2.1: Horizontally layered network architecture

For providing seamless high-quality services across different types of networks, the *network control layer* incorporates all the necessary functionalities. The network control layer houses a number of control servers that provide user session control and management functionalities. These control servers are also responsible for setting up and taking down packet sessions. The control layer also contains information about subscriber authentication, service authorization and localization.

We limit our discussion to the connectivity layer of layered network architecture. The reason is that our point of interest core network is a part of connectivity layer. In telecommunication network user session control and connectivity are separated into different layers. Asynchronous transfer mode (ATM) and internet protocol (IP) transmission are principle bases of connectivity layer. Connectivity layer is normally divided into two parts: *access network* and *core network*. Interfaces to legacy networks, such as the public switched telephone network (PSTN) are provided by connectivity layer. The layered architecture is being deployed in third-generation mobile networks that is, the universal mobile telecommunication system (UMTS).

Connectivity layer provides the transport mechanism for transporting any kind of information through voice, data and multimedia streams. Backbone architecture is

essentially made up of core and edge equipment. Core equipment is consists of backbone routers and backbone switches that handle traffic according to traffic engineering. Telecommunication operators predefine traffic route rules and traffic classification principles as traffic engineering. Core equipment transports and aggregates traffic streams between different terminal components at backbone edges. Edge equipment collects customer specific data and ensures QoS. The edge equipment is the telecommunication infrastructure nodes, the ultimate study focus of this thesis.

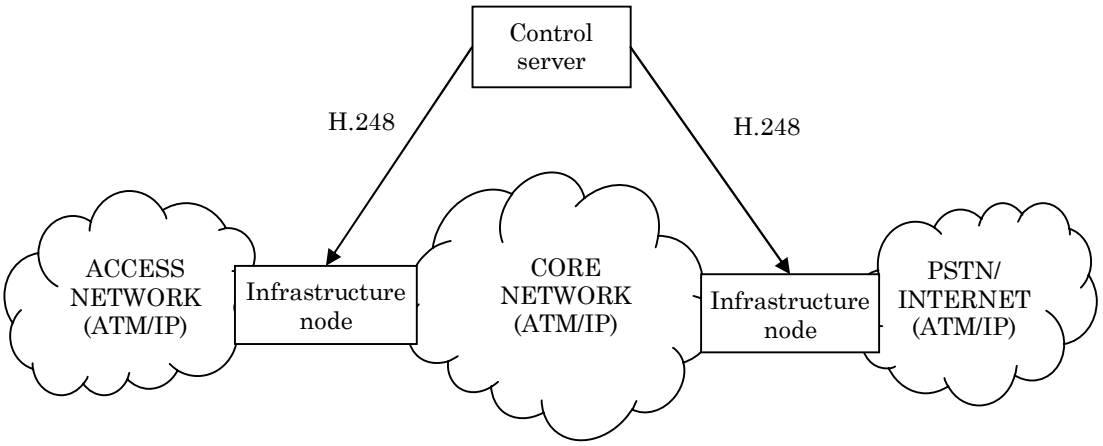

Figure 2.2: Basic connectivity layer structure represents core network

These nodes run under the full control of control layer nodes. These infrastructure nodes allows the processing of bit streams, and provide coding/decoding of speech streams, cancelling echo, bridging multi party calls and converting between transport protocols. Variety of services and applications executed by diverse network control domains can be accomplished by means of connectivity layer due to this exertion of control down to bit-stream level. Services and applications that are implemented and provided are truly autonomous of the applied transport technology. However the idea is that, this transport technology can vary over time due to evolving network but applications and services are never impacted.

2.1.1 Core network infrastructure nodes

Infrastructure nodes work within the core network architecture which is a part of a connectivity layer. They work as a bridge between different transmissions technologies to add services to end-user connections. One significant place of telecommunication nodes in the connectivity layer is at the bridging point between different networks. These nodes are also used in all-IP networks that support real-time voice-over IP. It can contain embedded real-time router [BLW99] and ATM/AAL2 switch with extensive

supports for quality of service and traffic engineering. These nodes mainly play two important roles: Firstly, switch ATM or route IP traffic and also provide internetworking functions between ATP and IP. Secondly, process the media stream (depending upon the service request from end user) and provide interfaces to different transmission technologies (such as PSTN). Telecommunication infrastructure nodes usually have a multi-rack, multi-blade architecture usually with specialized blades that are tailored for various functions (such as digital signal processing, packet handling, control software etc.). In practice these nodes usually contain tens to hundreds of processors of various types with thousands of cores interconnected using multiple networks with various topologies. In fact, a telecommunication infrastructure node is a computing cluster which is custom built to fulfill a specialized role. From architectural perspective, telecommunication infrastructure nodes are often built as hierarchical systems, where higher level software components (executing on one set of processors) control and manage lower level components (usually executing on another set of processors). Lower level components are usually handled as pools of resources and used by higher level components to full fill certain tasks (e.g. processing of user session). In such systems, setup of a user session normally broken down to smaller tasks handled by lower level software and hardware components. The hierarchical software architecture of the system also supports the non-uniform nature of the interconnected network between the cluster components.

The load in a telecommunication node varies greatly over time. In a capital city, for example, the load is likely to be the highest during office hours (perhaps peaking at near 100% of maximum capacity), whereas during the night and weekends the load becomes very low (well under 10% of maximum capacity); during long vacation periods the load can be even lower than that of peak hours.

2.2 Traffic handling in infrastructure nodes

As the telecommunication network has grown and advanced exponentially, the requirements of network interfaces have become more complex and diverse. These interfaces also serve complex purposes such as media processing, or any on-demand service. For serving various network applications and including multiple protocols requires highly intelligent and intensive traffic processing over the network. To keep up with current trends of emerging network applications, programmable microprocessors called network processors (NP) [LeJ03] are introduced in telecommunication core network interfaces to handle the demands of modern network

applications. For example, packet throughput of a 10Gbps link is 19.5 million packets per second, assuming a stream of minimum-sized packets of 64 bytes. Given a single processor of 1 GHz clock frequency, it can execute only 51 instructions per one packet time [LeJ03]. It is worth to mention that one single processor is not enough to perform the processing at wire rate. Hence, concurrent traffic handling through a suitable resource model is required to carry the traffic load where a number of processing equipment is involved (we will discuss this elaborately in chapter 3). Energy usage essentially increases while augments the number of processing components. This thesis is motivated to find the hypothetical way of possible energy optimization without compromising the network performance. There are two clear functional categorizations of NP applications in telecommunication nodes. These are two explicit traffic planes: control plane and data plane. User session establishment and call setup can be mentioned as the primary traffic generator activities in the core network. In principal, the control plane is responsible for managing traffic (e.g. call establishment, user data connection settlement). Data plane is responsible for processing the traffic as per session, or connection demand which is accomplished by reduced function application specific networked processor. We will discuss more in detail in following sections.

2.2.1 Control plane

The control plane of an infrastructure node is basically the set of processors that executes the higher level software components of the node. The traffic control plane is usually a cluster of commodity General purpose processors that are connected to form a HPC (high performance cluster) cluster. GPPs (General Purpose Processors) are flexible to rapidly develop network applications and protocols. They do not provide enough performance to process data at wire rates [LeJ03]. Therefore to process the traffic GPPs are connected to lower level data plane processing equipment (discussed in following section). The control plane is responsible for controlling the user sessions management and setup/release of calls/sessions requested by end-users. From the software deployment perspective, cluster front-end is configured differently than the other cluster components with necessary cluster management functionalities (such as load scheduling and storage). The performance of a telecommunication infrastructure node is measured by control plane functionalities. A number of metrics can be used for measuring the performance like following:

- Maximum number of simultaneous user sessions; this actually means the number of data connection e.g. in case of real time call conferencing.

- Maximum bandwidth; can be measured per user connection or in total.
- Session setup/tear down rate; this means data connection per unit of time.

The control plane is greatly responsible for fulfilling the above mentioned performance metrics in real time. Based on these metrics the network is pre-dimensioned to (i.e. the load threshold is determined) handle traffic. It is also worth mentioning that, traffic-load threshold can also be determined by calculating the traffic handling capacity of control plane cluster components and traffic processing capacity of the lower level networked processors. If the load level is over threshold, then the control plane gracefully discards traffic that cannot be handled reliably but still maintains the QoS of user sessions. The traffic load is distributed among the cluster components by the application software (deployed inside cluster front-end) through load distribution algorithm.

2.2.2 Data plane

This functional block is also known as a forwarding plane. The principle role of this functional block is to perform packet operations (e.g. forward data packets) and process the traffic at wire rate. The traffic data plane consists of programmable network processors, or dedicated ASICs (Application-Specific Integrated Circuit) [LeJ03]. These processors are reduced function processors tailored to perform traffic processing tasks (e.g. media stream dispensation). Control plane's equipment actually directs the demanded traffic processing work to data plane based on the user session requirement.

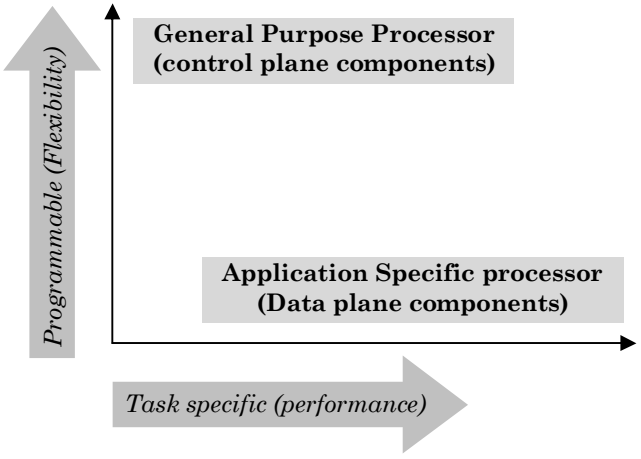

Figure 2.3: Performance vs. flexibility concept of NPs (network processors).

One or more data plane processing components can be connected to one control plane general purpose processing element. While processing traffic, a group of data plane processing components resides under a particular control plane component as a pool of

resources. The Figure 2.3 above represents a brief comparison between traffic plane processors in the telecommunication infrastructure node. The data plane actually performs packet processing for forwarding functions and any media processing on demand. Data plane functionality also includes traffic processing operations such as media transcoding and IP packet fragmentation. Media transcoding is a process of transforming media data object, e.g. conversion from one encoding to another. In case of packet fragmentation, data plane application splits IP packets into multiple fragments for which some header fields have to be adjusted and header checksum have to be computed. Data plane application also handles video mixing and tone handling for any video communication through mobile network.

2.2.3 Cluster based traffic operation

As discussed earlier, control plane is normally consists of commodity high performance general purpose processor units reside in a cluster. This cluster has a front-end (similar as other processor units within the cluster) which is different from rest of the processor units from system deployment perspective. Front-end processor units are deployed with all functionalities in the cluster with an extra functionality to manage cluster. In our system architecture, the front-end will also contain virtual machine management block. Every cluster component has its own embedded operating system running for traffic management activities and necessary protocol termination. Control plane cluster is different from any other cluster computing environment from both of the performance and operational perspectives. Every general purpose processing cluster component is connected with one or more application specific reduced function network processor(s). This cluster actually determines the node capacity of user session establishment rate. Every component establishes the user session by protocol termination and forward the data packet for processing toward data plane processing component.

In telecommunication infrastructure nodes, within the general purpose processor cluster, only the front-end has redundant GPP component to support failure recovery mechanism. The reason is that, the cluster front-end has all necessary information and managerial functional blocks regarding cluster such as load situation of all cluster components and load distribution algorithm. This redundant GPP resides within the cluster and connected in the same manner as the primary front-end unit for rest of the units and remains in switched-on state. Figure 2.4 below depicts the concept of control plane cluster system showing the redundancy of cluster head.

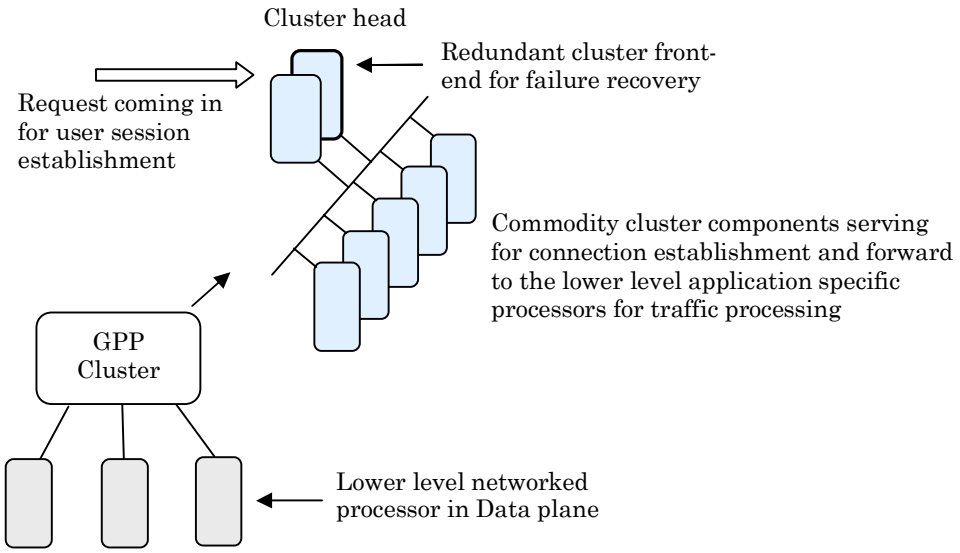

Figure 2.4: Control plane cluster

The Figure 2.5 below gives very basic energy usage scheme inside a telecommunication infrastructure node. It shows that, processing blades in different rack are the ones who are responsible for overall energy consumption by the whole node system.

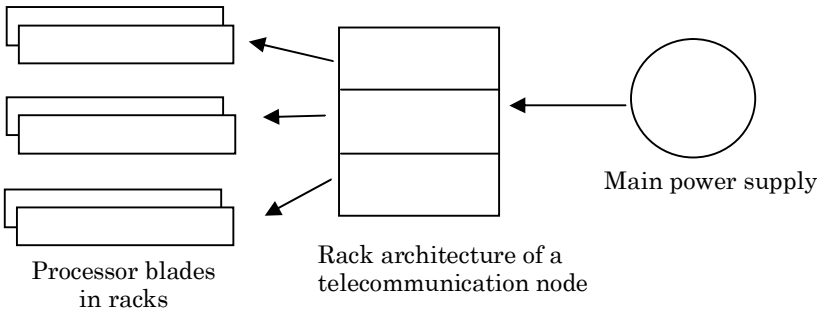

Figure 2.5: Basic power usage flow in multi-bade and multi-rack architecture

These processor components are the parts of multi-blade architecture which further creates multi rack node system. The Figure 2.5 above shows the simplified concept of power provisioning to the rack level energy consumption containing a number of processor components.

3 Traffic load scheduling inside infrastructure node

The basic idea of load scheduling in telecommunication infrastructure node is to distribute overall load as evenly as possible among available resources. This essentially minimizes the impact of hardware or software failures. In traditional system, for guaranteeing against hardware or software failures, all components were kept active all the time. This approach actually does not support power efficiency even at very low load levels. Energy-aware load scheduling allows loads to be concentrated to as few components as possible. This approach maintains the trade-off between in-service performance and energy saving. Switching off some resource components in low load level is the key for gaining energy efficiency.

3.1 Green factors

Both telecommunication network providers and operators are continuously making their effort to achieve green behavior from network activities. The goal is to establish efficient network to increase the overall carbon neutralization. Energy saving approach can be rooted into the operations performed by major network elements such as telecommunication infrastructure nodes. However huge amount of energy is used due to the operation of these infrastructure nodes in telecommunication network. The total amount of energy usage ramps up because of consuming energy in different levels of traffic handling. One watt possible saving by end level traffic operation has cascade effect on the overall node power consumption. In earlier discussions, we mentioned about the telecommunication nodes in mobile core network. In this section, we will study several green factors that can always influence telecommunication vendors to implement method to reduce power consumption inside those cluster-based nodes. The following four green factors for telecommunication infrastructure nodes are brought up in this study.

- Energy efficiency ratios
- Cooling requirements
- Space foot print
- Energy efficiency features

Calculating the energy efficiency ratios is significant. Current telecommunication nodes are like multi-rack based complex switching device where multiple blades are placed tailored for various functions (control or forwarding). Large telecommunication

nodes contain hundreds of processors of various types interconnected with various topologies. Energy efficiency ratio can be calculated per rack and per unit amount of data processing. For example we assume a node with n racks and each rack having m working blades process certain amount traffic per unit time. We assume the energy consumption measurement in the node per unit of time like following:

i	The index number of rack in multi-rack architecture.
j	The index of blade in a particular rack.
b_{ij}	Amount of traffic per unit time.

$$E_{ij} = f(b_{ij}) ; \quad (1)$$

$$E_t = \sum_{i=1}^{n} \sum_{j=1}^{m} E_{ij} ; \quad (2)$$

E_{ij} calculates the energy consumption in a particular blade for handling traffic per unit time. E_t measures the total energy usage in the node. For energy optimization we need to minimize the value of E_{ij} while b_{ij} is fluctuating depending on user demands.

Energy efficiency ratios directly related to the cooling requirements in the data center where nodes are placed. As long as the energy consumption increased in the switching node the heat production increased. Hence require a large number of cooling equipment which further consumes intensive amount of energy. Less energy usage in traffic operation will produce minimal heat from the nodes that reduce cooling requirements. Therefore we see that the second factor is directly related to the first one.

Now we move forward to mention briefly the factor concerning space foot print. This is beyond our study scope but for the sake of study we discuss this factor in brief. This is again related with the first one. If we develop the system in energy efficient manner then it will ensure the utilization of optimal resource in the node which will reduce the number of processing units due to possible optimization. Currently the telecommunication vendors are mainly heading to develop high capacity telecommunication nodes but they are a bit reluctant to look into the real optimal need of resources to serve the capacity targeted. Operators are engaging more nodes for escalating the core network capacity. As a result they are in need of more space requirements in data center to facilitate more physical systems. Space foot print also can involve new method of hardware design, which may ensure reduced amount of energy usage. But still developers can implement efficient software system to reduce energy usage [HJL+08] regardless of hardware design.

The last factor mentioned, actually regarding the features that can be implemented inside the traffic processing, or handling components to optimize the system not compromising with the capacity requirement but minimize the energy usage within that particular component. These features will make the system power-proportional [LWA+11]. Energy efficiency features will enable the whole system to behave seamlessly in elastic mood by dynamically scaling/shutting down some processing components. We see again this factor also has direct relation with the first one. The processor components of the control plane cluster are placed as means of processing blades in racks. Dynamic frequency scaling [TJL+08] is a good example of energy efficient behavior that can be deployed in embedded operating system environment. As we are concerned about the control plane cluster component for gaining energy efficient behavior, therefore elastic nature can be developed in system design to work inside every control plane cluster component (responsible ones for energy consumption ratio per rack). Elastic nature will put some blades in dormant mood which further minimizes the energy amount $E_{ij,}$ that eventually affects the total energy amount E_t consumed within the whole node system.

3.2 Resource model of cluster based system

Energy aware load-based scheduling method follows an abstract resource model. This resource model further divided into two significant model parts. Controlling resources at different levels is defined by *resource control model* and another one describes the responsible component that use particular resource named as *resource usage model*.

Resource control model: This model actually structures the computational cluster into hierarchical arrangement like tree. In a tree structure, two types of nodes are available, terminal and non-terminal node. Terminal nodes are the leaves whereas non-terminal nodes can have terminal nodes, or further non-terminal nodes under their possession. Now we will continue discussion about these two node types in case of resource control model.

In resource control model, terminal nodes are resources that essentially do not contain any other resources. Terminal nodes are processors, or processor cores that perform end traffic operations. These processors are able to work in energy saving mode that can be implemented in their operations. Non-terminal nodes are the owner of terminal resource components. In resource model structure, non-terminal resource components create sub-trees. This is typically a software component that can be co-located with other software components and also can be executed on a particular processor

component (e.g. GPP) as well. In cluster based infrastructure node, non-terminal node typically resides on the same processor component along with sub-tree. Hence, non-terminal nodes are called resource owners that manage lower level nodes. A resource owner contains the resource pool, which consists of non-terminal nodes. The upper most level in the tree structure contains redundant resource components. Redundant resource component is used to provide functional capabilities for having fault tolerant traffic operation inside the infrastructure node. Figure below gives general idea of this hierarchical structure with four levels of resources.

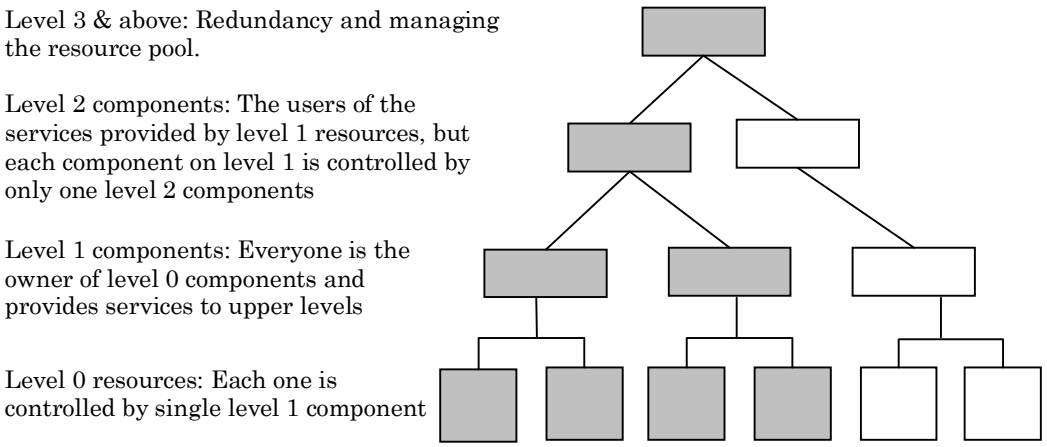

Figure 3.1: Resource control model in cluster-based node.

Terminal resource nodes are hierarchically managed. In the Figure 3.1 above, dark colored nodes are in active mode, whereas others are inactive and hence in power-save mode. Power-save mode encompasses several key rules to ensure energy efficient load scheduling. In operational state, at any particular time only active resources are used for processing. If the head node of any sub-tree is inactive, then all the nodes at lower levels are powered down for reducing energy usage. Lastly, in any situation, active status of a non-terminal node means that, at least one node (terminal or non-terminal) is active in the sub-tree. Non-terminal and terminal nodes are not always identical. Different resource pools can have different restrictions in processing.

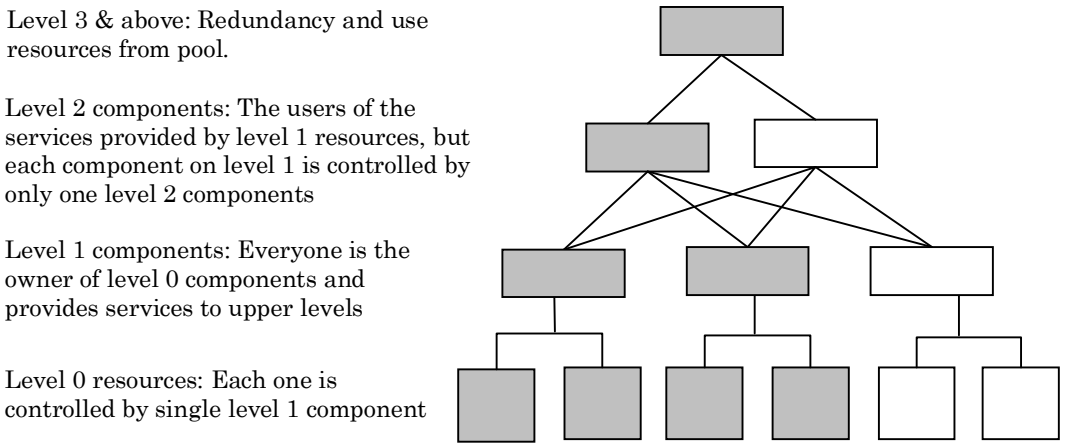

Figure 3.2: Resource usage model in cluster-based node

Resource usage model: This depicts the usage of resources. The ideology of resource usage model is similar but a bit different from resource control model. Two principle key points can be mentioned about resource usage model. Firstly, for any sub-tree there is a single access point to lower level resources, which is the non-terminal resource owner at the tree head. Secondly, any non-terminal resources at level *n* can use any non-terminal resources at level *n-1* (pools of resources). This ensures that pools of resources can be used from anywhere in the system at any particular time.

3.3 Load scheduling in cluster based node

Load scheduling method is a straight forward process, as it adjusts the amount of active resources online, subject to load situation of the system. To make this happen scheduling procedure relies on three significant metrics:

- highLoad
- lowLoad and,
- activeStep.

Firstly, *highLoad* is meant for the load level of the system when most of the resources are activated. According to observed load situation of the system *highLoad* parameter is pre-determined to serve the increasing load. *lowLoad* level determines that, some of the active resources can be turned into inactive state at certain point of time. Third metric *activeStep* is the crucial one, because it makes the system reliable by avoiding bouncing between low and high load states. To make the system reliable *activeStep* is calculated as the amount of resources (as absolute value, or as percentage of total capacity) and this is maintained while increasing/decreasing a pool of resources. This metric does not let the system to switch between low and high load states for arbitrary

set of active resources.

In the resource model (as discussed in earlier sections), every resource owner essentially has two sets of resources for making a resource pool. These are a set of active resources and a set of inactive resources. The first set is actually used for traffic operations. Later one consists of powered down resources. At the beginning of operation, *activeStep* amount (as mentioned above) of resources remains in active mode, keeping all other resources in the pool inactive.

Any resource owner at level *n* is responsible to manage the pool of resources at level *n-1*. Lower level active resources report their load situations (e.g. usage percentage) to their respective resource owners. Every resource owner uses the metric *aggregated load*, which is simply the average of load reported by the resources in the pool. With *N* number of resources in the pool the *aggregated load* is calculated as follows,

$$Aggregated\ load = \sum (Resource\ Load) / N$$

Load scheduling process works entirely on four different scenarios, where three of them are based on *aggregated load* and the last one is actually a decision making, impacted by three others.

- *Aggregated load* is between *lowLoad* and *highLoad* (normal operating mode).
- *Aggregated load* is over *highLoad.*
- *Aggregated load* is below *lowLoad.*
- *Deactivating a complete pool of resources.*

In normal operating mode, each of the active resources (terminal or non-terminal) always reports its current load status (for any particular load situation) to both owner and user of that resource. From this status report, resource owner knows about the resources (under its control) that are residing in active pool. When traffic operation continues, load status continually changes inside both of the *terminal* and *non-terminal resources*. In terminal resources, load situation changes due to new tasks, task completion, or any computational changes of ongoing tasks. For non-terminal resources, changes in load levels occur due to changed load status of lower level resources in active pool under its possession. Depending upon the load status report from lower level resources non-terminal resources calculate the *aggregated load* which is arithmetical average of loads reported by resources in its active pool.

Secondly, powered down inactive resources are activated and added to the active pool of a certain resource owner when the *aggregated load* crosses the *higLoad* threshold. A significant role is played by maintaining the *activeStep* amount of resources to prevent

bouncing situation as mentioned earlier. According to the *activeStep* measurement, when the *aggregated load* exceeds the threshold new additional resources are added in the pool. It is worth to mention that, if the resource pool is fully utilized by active resources (meaning that, no inactive set of resources found) then crossing the threshold does not trigger any action to add new resources. Activation of new resources can be depicted through the steps in the Figure 3.3 below.

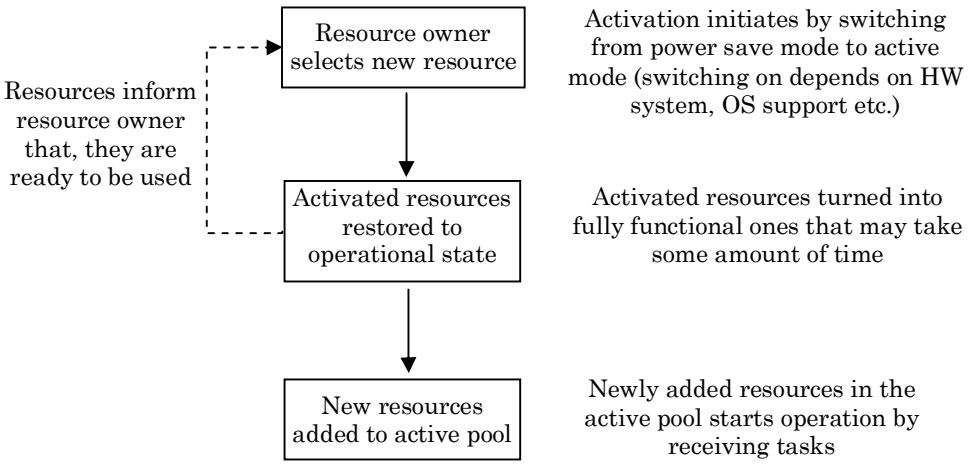

Figure 3.3: New resources activation

Resource owner selects any arbitrary resources from the pool depending on the load demand. For simplicity of the scheduling process, the way of switching on new resources is not discussed here. After the completion of steps mentioned in the Figure 3.3 above, resource owner reports new load level by calculating aggregated load and also measures the percentage of active resources in its resource pool.

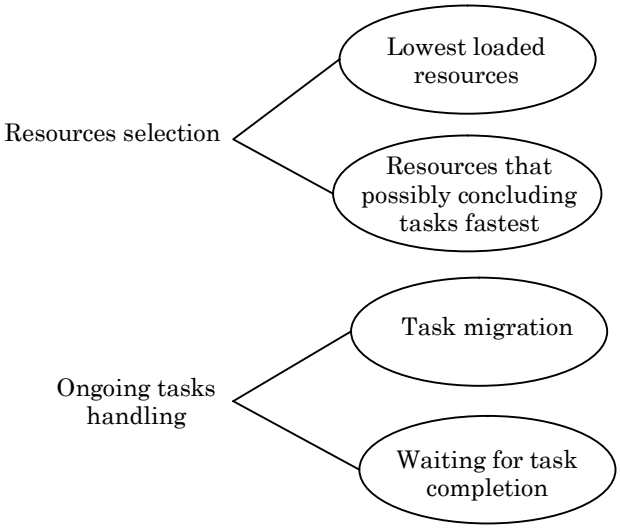

Figure 3.4: De-activating resources to power-save mode

Third case of load scheduling procedure is essentially to reduce the power consumption

by switching off resources. Like the previous step *activeStep* amount of resources is also maintained here. When the aggregated load is below the *lowLoad* threshold parameter, some resources are deactivated from the active pool. Two significant concerns are here: *selecting deactivation candidates* and *handling the ongoing tasks* in the selected resources. In the above Figure 3.4, we sketched the idea for resource selection and task handling progression while taking active resources to power-save mode. Selection process is done by resource owner. Two options are there for selecting active resources. According to load reports from lower level resources, resource owner can target the lowest loaded resources. The nature of tasks also impacts the selection decision. Along with the load reports, the resource owner also monitors the nature of tasks [SKV+12] that are ongoing in lower level resources. Sometimes it is beneficial to select resources where ongoing tasks are probable to be concluded faster comparing to tasks in lowest loaded resources. In both of the cases, de-activation is done after the normal completion of ongoing tasks. Once the selection is done, handling the ongoing task becomes the focal point. Firstly, the possibility of task migration is investigated so that, ongoing tasks can be migrated from selected resource to another active resource in the pool and original resource goes to power-save mode. In some situations, migration is not possible due to the nature of task e.g. user session would be terminated if migrated. In such cases, system waits until the ongoing task is terminated bounded by certain time and if time limit exceeds task is forcefully concluded.

Last scenario of load scheduling method is the decision by particular resource owner at a certain point of time, that it wants to de-activate a resource under its possession which itself is another resource owner. In this situation, request propagated down the resource management sub-tree owned by the targeted non-terminal resource. Upon getting the de-activation request it orders all non-terminal and terminal resources below to deactivate themselves. Whole deactivation process is done within a certain time period, if the time limit exceeds, forceful termination is done by principal decision making resource owner. Then lastly, resource-owner deactivates itself by reporting its users that, resource is no longer available to process traffic. Task migration does not take place in case of complete pool deactivation due to large number of processing task up to leaf non-terminal resources in the tree structure. After completion or termination of tasks, all non-terminal resources go to power save mode and propagate the load-report to upper level. Finally pool owner itself goes to power-save mode.

4 Virtualization and energy efficiency

Virtualization is an efficient method for taking advantage of modern computing hardware [ASR+10]. This actually improves the efficiency and availability of physical resources and applications by abstracting computing resources. This decouples system software from hardware resources. Virtualization technique can be divided into several categories depending on resource utilization demands, such as server/system virtualization, network virtualization, desktop virtualization, application virtualization. Virtualization technique provides significant benefit for resource utilization. In our discussion we will concentrate mainly on system virtualization technique. This effectively creates a new layer of software over underlying physical hardware called hypervisor (or virtual machine monitor). Hypervisor actually facilitates virtual machines to run with their operating and application services. This thesis study shows the possibility of deploying live migration technique (discussed later) of VMs to reduce energy usage especially in cluster-based telecommunication infrastructure nodes.

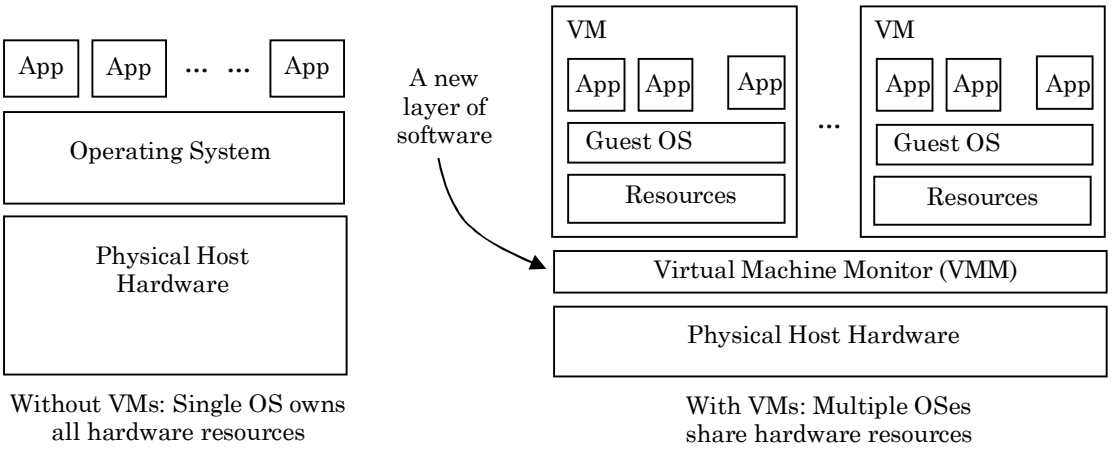

Figure 4.1: Virtual machine configuration (system virtualization)

The Figure 4.1 above depicts the general concept of system virtualization [VNE+08] as well as the virtualization of GPP components. Virtualization will assist control plane GPP cluster to run multiple VM instances in every single cluster component in low traffic situation through migration (discussed later). Virtualization of physical resources through VMM (virtual machine monitor) ensures better utilization of resources of a single cluster component up to its threshold processing capacity. System virtualization is gaining continuous research interest among operating system researchers worldwide.

4.1 Categories of virtual machines

A virtual machine is considered to be an efficient, isolated replica of the real machine because this has all the functionality of a physical machine with several significant properties. VM deployment leverages efficiency of the system as the unprivileged instructions are executed directly by the host processor, hence resource controlling becomes efficient. An application running in a VM cannot affect its own resource allowance without going through virtual machine monitor. There is actually no difference between the scenarios where a particular process 'executes inside a VM', or 'runs on real hardware'. Only exceptions are in case of timing and physical resource availability. In general, virtual machine monitor traps all instructions coming from applications that read, or write global machine state. This actually fools the application inside the VM believing that it alone controls the machine. Virtual machine discussion can be divided into several basic types.

System virtual machines: This can also be called as platform virtual machines. Hardware virtualization technique is used to create system virtual machines. Using this technique, in the same physical machine multiple OS environments can be co-allocated in strong isolation. Operating systems do not know about the proceedings running inside others. Virtual machine monitor hides the physical characteristics of a computing platform. System virtual machines are significant for providing several crucial properties to the whole system such as application provisioning, maintenance, high availability and disaster recovery. Application provisioning to users becomes easier due to system virtual machine deployment. As the application does not interact directly with the underlying hardware structure, hence execution of a particular application on heterogeneous platform becomes possible. As the unprivileged instructions can be executed so processes running inside VM do not see any difficulty while continuing operation.

In system virtualization technique, multiple virtual machines can run their own operating system which can be used in server consolidation. The idea of server consolidation is that, different services running in separate virtual machines on same physical machine avoiding intervention. System virtual machine concept leverages the facility to run different category operating systems on a single machine. Hence using virtual machine concept to support multiple guest operating systems is trendy in embedded system research.

Process virtual machines: In our thesis we opt not to study process VMs which are

called as application virtual machine. Inside host operating system this runs as normal application and supports single process. One example of this type VM is java virtual machine. These actually exist when the specific process started and dismissed after the process termination.

Pure virtual machines: The concept of pure virtual machine comes from the fact that, these virtual machines imitate the underlying hardware and also with the privileged executable instruction set.

When a machine is shared by several processes, it is important that proper nesting is observed. If a process is allowed to modify global state, such as the global interrupt disable flag, or the page table base register, this may affect how other processes execute, and may violate overall system integrity.

4.2 Virtual machine migration

This thesis focuses on the system virtualization concept (as discussed in earlier section) for operating traffic in the control plane cluster. In system virtualization technologies, VM migration is a key feature which is actually significant for increasing system's operational reliability from several perspectives such as failure recovery, resource availability etc. We have come across several research activities concentrating in this area [JDW+09] [CFH+05] [HiG09] [LJL+09]. The key concept behind virtual machine migration is that, this technique works by repositioning the memory and device state of virtual machine from one physical machine to another. Virtual machine mobility technique proffers motivating advantages for cluster computing environment that are serving applications through VM deployment. As the application running in the cluster component does not have direct physical resource utilization within a particular component, therefore several significant benefits can be seen by migration feature [LJL+09].

- Online load balancing
- Energy efficiency
- Transparent infrastructure maintenance

Online load balancing: We call it online because whenever any new request is coming in then proper load balancing among the components is necessary. To offer optimal handling of computing resources, virtual machines can be dynamically migrated to new physical machine.

Energy efficiency: VM mobility can reduce significant amount of energy in cluster

computing environment. Whole system can be squeezed depending on the load situation by consolidating to small number of physical machines. This behavior sets aside some cluster components by shutting/scaling down to reduce energy consumption in whole clustered system. Intensive research activities are underway to ensure most advantageous resource usage in data centers [LWA+11] to make the system energy proportional.

Transparent infrastructure maintenance: In a large cluster computing environment maintenance is obvious to ensure continuous reliability of the whole system. Moreover service interruption is not trivial, therefore system administrators can reposition virtual machine to other cluster components without any noticeable interruption of time critical [XHG+06] services. This benefit is not aligned with our study as we are not dealing with infrastructure maintenance. Still it is necessary to mention here as one of the significant advantages of migration.

Load intensive systems in most of the time face varying workloads. For performance reliability, a multi-processing-component based system works by keeping all the nodes in working mode. In some situations, due to varying workloads arbitrary number of nodes may be under-utilized, whereas others may be heavily-loaded. Therefore virtual machine migration approach can be worthy in such situations to balance VM loads among physical components of the system. Our study concerns with high performance computational cluster system. Cluster front-end periodically collects the resource usage statistics of cluster components. Based on this measurement some VMs are migrated to lower loaded physical cluster nodes. Energy efficiency is a critical research concern in modern multi-core, or multi-processor based computing systems. In these systems, by consolidating virtual machines to fewer physical machines it is possible to power off extra nodes to gain power efficiency.

4.3 Live migration of virtual machine

Successful live VM migration ensures that an application (that is running inside VM) does not experience any noticeable downtime in practice. Live virtual migration process can afford elastic way to use optimal power in high performance computing environment [NaS07] [HLM+09] [HJL+08].

Generally, live migration of entire virtual machine means on-line mobility of computational, or traffic processing server with minimal noticeable operational downtime [CFH+05]. Efficient migration of entire virtual machine essentially moves

VM's memory contents from source to destination physical host. Live migration is a powerful tool for consolidating clustered components into a single consistent host. Minimizing the perceptible operational interruption in live migration is possible based on two significant metrics: *downtime* and *total migration time*. *Downtime* is the actual service interruption time which means that control functionalities (in control plane cluster) will be stopped as no VM is running. The *total migration time* is the sum of downtime and actual migration time. The *total migration time* is the sum of downtime and actual migration time. Actual migration time is the time period that is measured from migration initiation to activation of VM in another physical machine. Any computational overhead due to migration process increases the migration time which impacts the *total migration time*. Moreover *total migration time* also increases with longer *Downtime*. During this time period memory contents of particular VM is fully transferred to another physical machine. The method of transferring VM's memory pages is the most significant concern in live virtual machine migration. Efficient transferring method may ensure minimizing the above mentioned timing metrics. Memory transfer method can be depicted by dividing into the following three steps [CFH+05].

Push phase – During the migration process memory pages of running VM is pushed to new destination. As the source VM keeps on executing all application services running, so memory that are pushed to new destination may be dirtied during ongoing migration process. These pages need to be pushed again, or re-sent to new destinations.

Stop-and-copy phase – In this step, VM on the source physical machine is clogged for a while and pages are copied across to the destination VM. This can be considered as the final step of migration process. After this phase source VM is terminated and new VM starts serving. Service interruption occurs in this step. Time required in this phase is essentially called the *downtime* seen by the ongoing functionalities inside VM.

Pull phase – This phase shows up just before the termination of source VM. After the new VM starts execution, if it accesses a page that has not yet been copied then this page is faulted in (pulled) across the network from the source VM. Once new VM executes reliably with all functionalities, then old VM is terminated. Functional verification of the migrated applications (that were running inside the source VM) is not explored in this thesis and kept out scoped. It can be well assumed that, if pull phase prolongs then *total migration time* may increase.

Several approaches of virtual machine migration are investigated by researchers [PKC+09] [MoC10] [LJL+11] [HGW+11] [HiG09]. Post-copy [HiG09] and pre-copy

[CFH+05] are two significant migration techniques that are worth mentioning here. In the following sections we will discuss these two types of live virtual machine migration technique.

4.3.1 Post-copy migration

During the initiation of post-copy migration, virtual machine at the source physical host is suspended. After the suspension of virtual machine, the smallest subset of the execution state (such as CPU registers and non-pageable memory) of virtual machine is migrated to the target physical host. When all necessary CPU states are transferred to the destination physical host, then the new virtual machine is started on the destination host even though major part of the memory state of the virtual machine still resides at the source physical host. After that all memory pages are transferred from source to destination using the push-phase. On a high-level, post-copy migration defers the memory transfer phase until after the virtual machine's CPU state has already been transferred to the target and resumed there [HiG09]. At the target host, virtual machine generates page-faults when it tries to access memory pages that are not yet transferred. These faulted memory-pages are demand-paged over the network from the source. In post-copy migration, this process ensures that each memory page is transferred at most once. But on the other hand, because of redirecting each page fault (of the running virtual machine) towards the source can mortify the performance of application running inside the virtual machine. Pre-copy migration technique is free from this performance bottleneck of the running application inside the virtual machine. In the following sections we will discuss more in detail.

4.3.2 Pre-copy migration

Pre-copy approach uses iterative cloning of memory pages in rounds. In every second round, the pages that are dirtied during previous copy round will be copied again to the destination VM. This process continues up to finding a writable working set (WWS) [MLL10]. Continuous cloning of VM's memory pages results into small set of application's writable working set (WWS). Memory page dirty rate may not be the same for all ongoing functionalities. Once the WWS is found, source VM is suspended and all CPU states and remaining dirty pages are copied to destination. According to several research studies [DYS+10] [HJL+08] [OSS+02], virtual machine migration techniques mostly use pre-copy migration, as VM services are always available during migration process except short stop-and-copy phase. But some crucial concerns can be

mentioned here. Additional computational [SZL+11] and network overhead [PiY10] can be traced out due to iterative copy method. Maximum number of iterations can be set to come up with writable working set. This is significant, because it might be the case that we may never converge to a small set of idle memory pages due to continuous page dirtying by different applications. Network overhead is another crucial concern. Iterative cloning approach may acquire most of the bandwidth inside the cluster. Therefore, pre-copy migration method is going under intensive research [LJL+11] [MoC10] to show optimal performance.

In this thesis we opt to study the pre-copy virtual machine migration approach to balance trade-offs among above mentioned constraints (such as computational overhead, network overhead) and migration phases (e.g. writable working set). It differs from post-copy process, as in this case CPU states are copied after copying all memory pages to destination and then starts the destination VM.

4.4 Downtime during migration process

The *downtime* observed during the migration process essentially causes degraded quality of services. Minimizing both the *downtime* and *total migration time* is always challenging. Both *downtime* and *total migration time* are relative to the amount of physical memory allocated to the VM. Efficient migration process needs to minimize the both. Minimal downtime can impact on service interruption and also least possible *total migration time*. In general, we need shortest possible stop-and-copy and pull phases. The *downtime* is a period of time when no CPU cycle is engaged to serve any of the applications that are facilitated by the source or destination VM. The *downtime* is actually the sum of several different time slots [MLL10]: the time required for suspending the source VM, transferring the VM state to the destination, loading the VM state to the destination and activating the migrated VM on the remote physical host. In pre-copy migration approach *downtime* is minimized due to iterative copy operations, because cloning process continues up to finding out WWS. Iterative copying technique can make stop-and-copy phase much shorter. Researchers showed that the very straight-forward way of virtual machine migration is pure stop-and-copy approach [WCS+04]. In this method, original VM halts and then entire memory is copied to the destination. This essentially reduces the total migration time but higher down-time is observed. Additionally another approach pure on-demand [Zay87] migration suffers from high total migration time. Pre-copy migration [CFH+05] is free from these problems as it iteratively copies the memory pages to destination host. And lastly faces

a very minimal stop-and-copy phase.

4.5 Pre-copy VM migration in GPP cluster

This thesis explores the possibility of applying pre-copy virtual machine migration technique in control plane cluster environment. After virtualizing the identical general purpose processor components, every cluster component contains one virtual machine in it to perform traffic control functionalities. In this study we show that the pre-copy migration approach can be applied to consolidate virtual machines in fewer physical components in low traffic situation. This consolidation can help the system to power down extra cluster components for reducing energy consumption. The core idea of this method is that, upon the direction from migration daemon in the cluster front-end, source and destination GPPs are determined. Then after that following stages [MLL10] are performed for the completion of migration process:

- Initialization: This is the starting phase when migration starts up. Target GPP is selected by migration daemon and both of the source and destination VMMs are notified.
- Reservation: After getting the notification, destination VMM reserves sufficient resources for incoming GPP virtual machine.
- Iterative pre-copy: This phase is actual live migration phase where memory pages are iteratively copied to destination GPP.
- Stop-and-copy: After copying most of memory pages this is the final round to stop the original VM and then rest of the pages are copied to the destination.
- Commitment: When the stop-and-copy phase is completed, the destination GPP component acknowledges that it has received everything of the source VM.
- Activation: New VM is activated and continue the functionality and source VM is terminated by hypervisor.

In our hypothetical system, we need to maintain at least the legacy throughput of the control plane. Therefore it is also necessary to minimize the extra computational overhead in cluster due to migration that may affect control plane performance. To turn this into action, we need to consider the following:

Total-Migration-Time, t_1; we need to minimize

Total-Down-Time, t_2; we need to minimize, as this affects total migration time and also

during this time period user will face service outage.

t_1 = Initialization + reservation + ∑pre-copy + stop-and-copy + commitment + activation

t_2 = stop-and-copy + commitment + activation

Here we see that t_1 and t_2 do not have direct relationship with the energy usage in the cluster. But to ensure successful virtual machine migration we need to calculate those. These are significant because we opt for VM migration to gain energy efficiency and at the same time we do not want to face any performance degradation due to virtual machine migration, or energy optimization. The Figure 4.2 below shows the concept of iterative pre-copy approach of migration [MLL10].

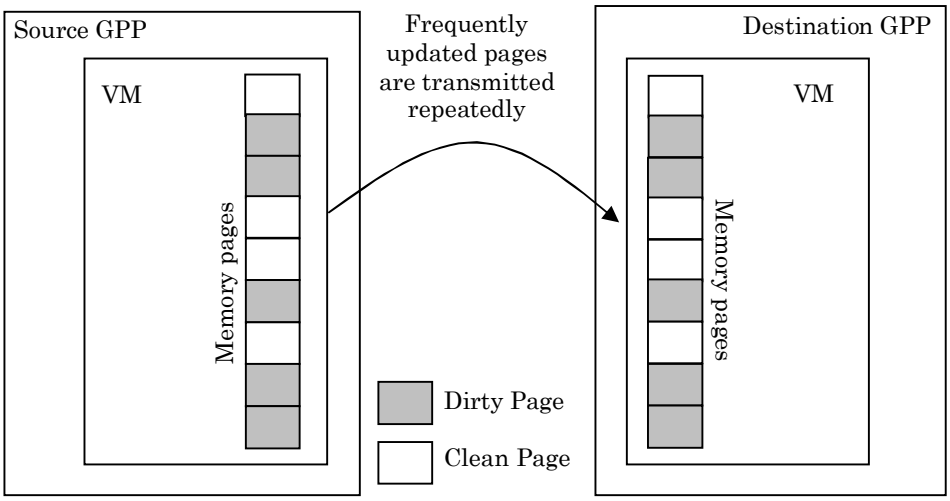

Figure 4.2: Iterations in pre-copy live migration

As we studied that the iterative copy approach in live migration works by copying dirtied memory pages continuously to the destination [ASR+10], so one significant point is to determine when to stop copying and proceed to final stop-and-copy phase. Therefore the maximum number of repetitions of the copy process (in the iterative pre-copy phase) needs to be decided dynamically or pre-determined. Stopping condition of iterative copying approach can be determined by identifying a small writable working set [CFH+05], or a preset number of iterations. We mentioned earlier that telecommunication infrastructure nodes are the load intensive equipment in the core network. So in this case WWS (writable working set) may not pledge to congregate across successive iterations. Hence determining the maximum number of repetitions in the iterative copy phase could be beneficial in this case. This means that we can have a pre-defined number of iterations during the execution of the migration process.

4.6 VM mobility to reduce energy consumption

In this thesis we use the term “elasticity” for VM mobility as the process to obtain energy efficiency in control plane cluster. VM movement actually gives the elastic capability in cluster which reduces the cluster size in low load situation. Our study is not concerned about cloud computing approach but still we are proceeding with very basic thought of cloud computing i.e. virtualization. We are concerned about the system virtualization of individual component rather than whole infrastructure virtualization.

We notice this elastic approach a bit differently from Amazon EC2 concept where the technique is to provide supplementary elastic aptitude to the virtual infrastructure for the augmented service demands, or to gratify climax demand periods. Our VM migration technique will dynamically redistribute the virtual machines to fewer physical hosts during the low load situation to power down some physical components to reduce energy usage. This nature will shrink the cluster size in low traffic condition and we call it ‘squeezing’ situation. Again during high load level in a single cluster component some inactive physical hosts will be activated. VMs from highly loaded hosts will move back to newly activated hosts and hence the cluster size will be expanded which defines the ‘releasing’ situation. Literally ‘squeezing’ and ‘releasing’ behavior of the system (based on load situation) can be observed as elastic nature in the control plane cluster. Therefore elasticity works for gaining energy efficiency within the cluster. In the next chapter we will discuss more in detail about live virtual machine mobility through migration technique.

5 Reducing energy usage in control plane

The aim of this thesis is to reduce energy consumption in traffic operations inside telecommunication infrastructure nodes. The study of this thesis is mainly focused on the possibility of reducing energy usage in the control plane (consists of clustered GPP components) traffic operations. This chapter will discuss about the virtual machine migration opportunity in high performance computational control plane cluster. We will study the virtualization of clustered resource components, architecture of energy awareness, virtualization scopes (in several sections), and finally depict an optimization model of virtual machine mobility for gaining elasticity in the control plane GPP cluster. We will discuss the overall possible design and working concept of live virtual machine migration in the control plane cluster. We illustrate the hypothetical way of *squeezing* and *releasing* the cluster size through VM migration that can provide elasticity within the cluster. The study goal of this chapter is to show that it is possible to reduce energy usage especially in low traffic situation and also to use optimal energy during high traffic situation.

5.1 Limitation of control plane load scheduling

Studies and discussions in previous chapters and sections have depicted the resource model architecture (discussed in chapter 3) for traffic operation in cluster-based telecommunication infrastructure nodes. Traffic scheduling technique in cluster based resource model is not sufficient to achieve energy efficiency from higher level general purpose processors. Because they are connected to a number of application specific network processors of data plane. Therefore energy consumption due to control plane functionalities in infrastructure nodes currently is not optimal. According to this scheduling technique, computing resources reside in a pool and from the pool optimal number of processing components are waken up for traffic processing. A number of resources can be powered down after a certain idle period of time that is pre-determined depending on user session establishment request coming in with traffic processing demands. The following discussion will highlight the significant points for which the energy efficient resource scheduling is not optimal due to end level traffic processing in data plane.

Power-aware resource scheduling mechanism cannot be used in control plane traffic management. The reasons can be pointed out as follows:

- GPPs are connected to multiple ASPs in levels.
- Calculating idle time is not straightforward (may be problematic).

Firstly, multiple data plane application specific processors are connected to any particular processor component of control plane cluster. As the control plane cluster is made of higher level processor components in the resource model, therefore scale/shut down the GPP components is not straight forward. The reason is that, non-terminal GPP components are connected to another non-terminal components, or lower level application specific processors in data planes. All the non-terminal higher level processor components need to be in powered-on state up to the completion of processing task by any single end level application specific processor component.

Secondly, if however GPP components are considered as a pool of processing components it would be very troublesome to get any certain idle period for a particular processor. This is because of having tree resource model with multiple levels. We consider a situation when GPP components are not managing any traffic during low load. But still non-terminal processing components could continue processing traffic of previous user data connections. Therefore, as these resources are connected to higher level GPPs, so in these cases GPPs cannot be powered down.

To achieve power-aware behavior in control plane traffic management we are motivated to study the possibility of deploying virtual machines inside GPP components through system virtualization technique. Hence the live virtual machine migration process can be the doable solution to reduce energy consumption. VM migration will facilitate the mobility of entire OS (running inside every control plane cluster component) among GPPs that could keep lower level processing tasks noticeably uninterrupted. Lower level terminal processing components in data plane do not see any change happening in upper level GPPs and continue processing traffic alike as before migration process. VM consolidation provides facility to switch off a particular GPP component. After successful migration the source cluster component is powered down.

5.2 Virtualization of cluster component

Typically virtualization process provides virtualized hardware interfaces to VMs through a virtual machine monitor (VMM), also called hypervisor. Implementing VM technology in control plane cluster allows running different guest VMs in a single GPP component. Each guest (migrated in our case) VM runs its own operating system. But

still virtualization facing some challenges like computational overhead due to virtualization. Memory consumption of the system is also significant for virtualization, because this technique allows multiple virtual machines to use single physical hardware. In the following section we study briefly about resource provisioning through virtual machine monitor, or hypervisor.

5.2.1 Hypervisor and resource provisioning

In this subsection we study the possible new software layer of virtual machine monitor or hypervisor in GPP components. The use of hypervisor provides the facility to deploy application runtime environment on-demand over dynamic computing resources [LHL+08]. We discuss about the hypervisor in system virtualization environment. From the theoretical discussion in previous chapter we see that, hypervisor or VMM directly resides on the physical hardware. Significant issues are there, as we are not discussing about hosted virtualization. In hosted virtualization environment the hypervisor is implemented under the hosted operating system. OS directly interacts with underlying physical hardware and VMM interacts through the hosted OS. This creates additional computational overhead due to extra layer of VMM on top of hosted OS. Moreover in our hypothetical system structure we opt for switching off particular cluster component for energy efficiency. If hosted virtualization is used, then it would be challenging to squeeze the cluster size through migration. Because in that case, hosted operating system may need to remain in running state always to make interaction with VMM.

System virtualization technique is free from additional operational overhead as in this case VMM resides on the physical hardware (as shown in the Figure 3.7). So if no virtual machine running over it then it can be switched off, or scaled down. Hence, system-level virtualization in GPP cluster can be used to provide a set of system services, such as migration, suspension/resumption and also termination of running virtual machines. These are necessary for our power-aware system structure. In system level virtualization, hypervisor stores a large dataset in memory such as memory map of virtual machines. Virtual machine monitor keeps a map of VM's memory spaces [CFH+05], or at least the page tables. Therefore VMM potentially can have a large memory footprint. This has direct impact on VM migration time (will be discussed in later section). As virtual machines are isolated from the physical platform; hence in this case hypervisor can be used to take care of VM execution and termination along with migration facilities. Hypervisor will work in connection with migration

daemon residing inside cluster head end and act upon migration decision made by daemon.

Privileged operations from the guest VMs are trapped and processed by Virtual machine monitor (VMM). Resources in every single GPP cluster component are virtualized for being used by one or multiple virtual machines at any particular time. While running multiple VMs, online allocation of computational resources is essential to the application services when requested. We studied dynamic allocation as the resource virtualization which is free from over-provisioning and under provisioning [CLN10]. Dynamic allocation does not partition the underlying infrastructure resources to specific application services running inside virtual machines. VMM simply works as resource exposure to the virtual machines running on it through possible VMM-bypass [HLA+06]. VMs are free to use the resources as much as needed up to the overall threshold processing capacity of particular physical GPP component. We study later that hypervisor can keep load statistics of its own GPP component to report to cluster manger. If the load level is about to reach the threshold then hypervisor informs the cluster manager for making migration decision.

5.3 Energy aware control plane architecture

In this section, we discuss the possible system architecture that can ensure energy efficient behavior of control plane cluster. The Figure 5.1 below shows the possible system structure having migration facility of VMs, where cluster front-end performs the major role.

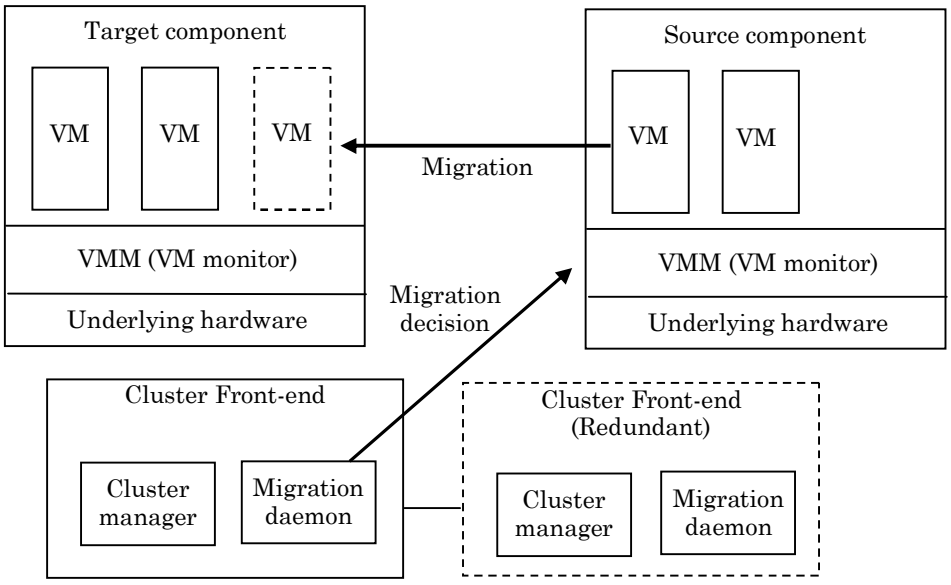

Figure 5.1: Energy-aware control plane system architecture

In low traffic situation, migration decision is made by a software component implemented inside the cluster front-end called migration daemon. When the migration is done then the source component goes to sleep, or shut down. Every physical cluster component always reports its load situation to cluster manger. Operational loads of VMs are monitored by virtual machine monitor (VMM) in every GPP component. Load statistics of virtual machines are used in redistribution of VMs in fewer physical machines for switching off rest of the GPP components to reduce energy consumption.

Power-aware behavior depends on two load states:

- High traffic load
- Low load situation

In our system, high traffic does not mean the overall load of physical components in the cluster. There is a high load threshold pre-determined and same for every cluster component. So whenever any particular cluster component is about to exceed the threshold then cluster front-end (through migration daemon) initiate migration. According to migration decision one, or more VMs (located on that physical GPP component) are migrated to low loaded physical components. Here low loaded component means also the newly waken up physical one which was previously shut down. As the cluster front-end has the load statistics of all physical components, so it selects the destination host which has sufficient physical resources to facilitate the incoming VM(s) and directs migration daemon to perform migration. In case of having multiple destination hosts possessing same capability, then selection can be random. This random selection can be performed for both ways of elasticity i.e. *squeezing* and *releasing* the cluster size. One significant point is that for squeezing the cluster, front-end uses the load statistics of the whole cluster, so that redistribution of VMs can be performed to minimal number of physical components as possible. On the other hand, for releasing the cluster size load situation of particular component is considered. It is worth mentioning that, initialization and reservation of resources to facilitate a VM in the target component can be done instantly when selected. Then copying the memory pages, CPU states, commitment and activation phases of VM migration are done.

5.4 Data plane traffic processing

According to the resource model of cluster based infrastructure node, non-terminal GPP components are responsible for traffic management (e.g. user sessions, call control) whereas lower level data plane processors perform the end level traffic

processing functions. In the resource model of cluster based infrastructure node, the control plane GPP cluster components are the top level non-terminal nodes in the structure. A non-terminal node can be connected to one, or more non-terminal, or terminal nodes in lower levels. These lower level terminal nodes are essentially the application specific reduced function processor of data plane.

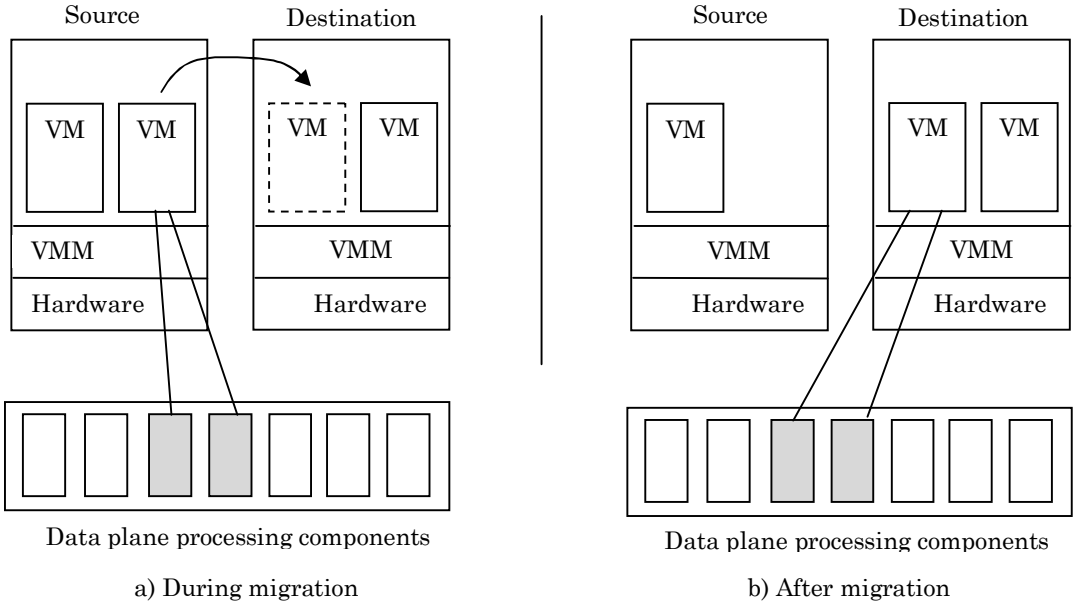

Figure 5.2: Data plane connection with virtualized GPP components

The study of this thesis is concerned about the control plane traffic operations inside the cluster-based telecommunication node. Data plane traffic processing operations are out scoped in this study. But still the reason of mentioning data plane discussion here is to confer the working concept in connection with VM based control plane cluster. In current situation data plane processors have direct connection with control plane processor components. In this possible new system architecture data plane processors will be connected with VMs of clustered GPP components. Application specific processors (in data plane) may not face any noticeable interruptions in their operations due to the mobility of VMs among physical components. This can be achieved by considering the complete application context [SZL+11] running on top of the VM when choosing the appropriate general purpose processor component to host that virtual machine. This will help to incorporate inter-VM dependencies and the underlying network topology into VM migration decisions. The goal is, not to impact traffic processing in data plane due to virtual machine migration process. The Figure 5.2 shown above makes it easier to understand the concept.

5.5 Scheduling technique among virtual machines

We discuss possible scheduling technique in this thesis for distributing traffic among the VMs of cluster components. In our system structure, scheduling process performed by the cluster manger inside cluster front-end. Traffic scheduling occurs among woken up and active physical components that are running one, or more virtual machines. Scheduling algorithm works in connection with traffic load situation of cluster components. Migration daemon inside the cluster front-end is mainly responsible for initiating virtual machine migration. Optimal scheduling of traffic is significant to achieve best possible migration performance. We outline following steps for traffic scheduling in our hypothetical cluster system.

- High load threshold is maintained for every physical component. We use traditional naming i.e. *highLoad* for this parameter. High load threshold value is same for every component in the cluster.
- Cluster manger aware of resource usage of every GPP component. For a new user session request coming in, cluster manger checks for particular GPP with available resources. Cluster manager selects physical component based on pre-defined value of *highLoad* parameter.
- Controlling service request goes to selected cluster component. Virtual machine monitor receives the request. If multiple virtual machines running inside that component, then hypervisor selects any of the VMs to serve. VMM can select based on lowest loaded VM or any VM that running fastest completion possible tasks.

Cluster manager periodically gets load situation of particular cluster component. In our system architecture cluster manager maintains a list of all virtual machines of cluster components. This new scheduling process is different from the state-of-the-art technology. Previously three parameters *highLoad*, *lowLoad* and *aggregated load* were determined to ensure optimal scheduling among the resources in tree-based model. Optimal resources were kept in active mode to gain energy efficiency, while keeping rest of the resources inactive and powered down. In our cluster system, traffic scheduling is necessary due to ensure efficient resource usage of GPP components. The decision to power down a particular physical GPP component is made by virtual machine migration process. Therefore *lowLoad* and *aggregated load* are not taken into consideration for scheduling decision.

5.6 VM migration in GPP cluster

Control plane GPP cluster as mentioned earlier consists of commodity processing units interconnected by high-speed and low-latency network. We can never compromise with the cluster performance for reducing energy. So there is a trade-off between maintaining in-service performance and energy saving within the cluster. In this section we find out the appropriate live virtual machine migration technology for control plane GPP cluster of a telecommunication node. To apply VM migration we analyze two migration techniques. Stop-and-copy approach and iterative pre-copy migration approach. First we analyze better possible pre-copy migration technique. Then after that we analyze the suitable migration approach for the GPP control plane cluster. VM migration in control plane GPP cluster is slightly different from traditional approaches due to the number of virtual and physical machines. In traditional live VM migration technique in most cases the number of VMs is larger than the number of physical machines. This ensures better resource utilization by redistributing the virtual machines. The basic difference in this high performance control plane cluster is that, the numbers of virtual and physical machines are same. This means, always the number of VMs remains static regardless of low, or high traffic situation. We opt for incorporating VM migration to reduce energy usage in different load situations as we don't need all the physical machines to up and running in low traffic situation. So according to our goal, in low traffic situation some VMs migrate from their original physical host to new physical host within the cluster that has enough resource available to facilitate them (migratory VMs). High load and low load thresholds are determined according to SLA (Service Level Agreement). Load levels are calculated for every physical host within the cluster by cluster front-end which is also consists of same hardware but with different configuration.

To Migrate the VM in this high performance GPP cluster we take several metrics in consideration. These are crucial for live VM migration in control plane cluster: *Total migration time*, *total data transmitted*. The study of this thesis opts for reducing energy usage through VM migration but in this case VM migration depends on these metrics mainly because of high performance requirements in the cluster. Our study is based on several existing research material in live VM migration area.

Every GPP components in the cluster will contain VM to serve traffic application in the control plane. Live VM migration takes the running VM from a certain cluster component and place it to another physical component and shut/scale down the first one. Simultaneously while looking to optimize energy usage, we also need to maintain

good performance of VM migration technique. Following are the metrics that might weigh successful VM migration in cluster.

Total migration time: Total time can be calculated which is required from initiating migration to activation of new VM in another physical machine. Live migration itself follows several steps to complete. Initialization, resource reservation, copying VM memory pages, acknowledgement and activation of newly copied VM. Copying VM memory pages is further divided in iterative copy process and final copy iteration.

Downtime: Actually this can be considered as a part of *total migration time* (mentioned in section 3.4.3). During this time VM in the source cluster component stops, meaning that no CPU cycle serves for occupier applications at source physical component. Then VM states transfer to the targeted component and activation of new VM on the targeted component is done. The *downtime* in total is the measurement of suspending VM on the source component, transferring VM states to target physical component and activating the new VM on the destination.

This study discusses the HPC (high performance computing) control plane cluster for applying VM migration. So the probable computational overhead of VM migration is crucial as it may affect the traffic management functionality by consuming link speed and through processing overhead. If we opt for iterative memory copying approach [PKC+09] [MoC10] [PiY10] [LJL+11] [SZL+11] to facilitate live migration we are likely to face above mentioned overheads.

After having the method chosen for migration now we depict the possible way of working for choosing particular virtual machine to migrate to a new physical host within the cluster. In this hypothetical system, hypervisor knows which one is the migrated OS as all the migrated ones are the guest OS in the destination machine.

In our study we discuss that migration can take place in both of the high and low traffic situations. This makes the VM migration in this study different from any other state-of-the-art VM migration technology research. As we gone through a number of research articles of VM migration, we observed that the technology is not applied particularly according to traffic load situation. Migration technique applied for proper load balancing among physical hosts and maintenance of any particular cluster component [NaS07]. These ideologies give some benefits to reduce energy usage [HJL+08] but our strategy is truly targeted to energy optimization. At the same time we investigate the better way for migrating VM, while minimizing the *downtime* due to migration.

In managed migration all the memory pages are transferred to the destination VM. In subsequent rounds it will send the pages that are dirtied during the 1st round and so forth. So in this way all the memory pages would be transferred. As a whole in this section, we study the apposite way of VM migration in this sort of high computing cluster. For measuring the suitable migration technique, we need to do some investigation work that fits the need. Now the goal is to find out the specific metrics for proving appropriateness.

For successful VM migration we need to consider the compute cost in total:

We assume the cluster operation for a period of time T,

Energy used in normal operation without VM migration, E_1

Total energy usage with VM migration, E_2

Energy usage reduction = $E_1(T) - E_2(T)$

This is a very straight forward measurement of energy usage reduction. But this entirely depends on the successful VM migration through optimized migration decisions. In following sections we study the hypothetical way of decision making for *squeezing* and *releasing* the cluster size.

5.7 Optimization model of migration decision

Ideology behind the power saving in a telecommunication infrastructure node essentially is to balance overall load among fewer number of physical cluster nodes. As discussed in chapter 3, energy efficiency can be gained by making a pool of lower level resources in resource control model. This feature can reduce the energy usage in network switching node. Our studied method is not following the strategy alike but the same switching on/off technique. The hypothetical system will shut/scale down a cluster component through virtual machine migration. This can be done by redistributing the VMs (through live migration) among optimal number of physical components. Redistribution through migration of virtual machine will free up certain cluster component. That particular cluster component will shut/scale down. This mechanism will apparently squeeze and release the system dynamically depending on traffic load situation. Gaining energy reduction by switching off actually depends on several metrics.

- Overall load situation of the cluster; essential for taking migration decision.
- Particular resource usage statistics of every physical component; whether it can

facilitate certain VM(s), or needs to offload VM(s).

- Load statistics of every single virtual machine.

Achieving optimal effect is quite straight forward. Firstly, calculate the overall load of the system and being divided by the capacity of a single physical machine to get the number of nodes which can handle the overall load. Secondly, transfer the scattered load to the calculated minimal set of physical components by a sequence of live migrations. The challenge is to obtain the optimal effect and also decrease the overhead caused by VM migrations.

Optimal VM mobility among the physical machines apparently will ensure the reduced energy usage in the cluster. Optimal mobility is significant to avoid extra computational, or network overhead due to VM migration. We discuss potential energy usage reduction through an optimization model of VM migration decision. For cluster we assume a common threshold of processor capacity level of particular cluster component for serving control plane functionality. Optimization problem is targeted to determine the minimal number of physical cluster GPP components that are capable to serve the current load situation of the cluster. To solve this problem we investigated the multi-capacity bin packing problem [LKK99] and come up with the following statement:

C_T	Threshold processing capacity of every physical GPP.
n	Total number of virtual machines.
d	Total number of physical machine working at migration startup.
C_k	Available capacity of physical GPP k at migration startup.
w_i	Processing capacity needed by virtual machine i.
B	Minimal number of physical GPP required for facilitating n virtual machines.

We need to find minimal B in a form that $B = \{b_1, b_2, b_3, \ldots, b_m\}$ where each b_j is the set of VMs placed in a single GPP cluster component k and is subject to the following constraints:

$$w_i \leqslant C_k \qquad (1)$$

$$C_k \geqslant min\{w_1, w_2, \ldots, w_n\} \qquad (2)$$

$$i \in b_j \;;\; 1 \leqslant j \leqslant m, \;\; 1 \leqslant i \leqslant n \qquad (3)$$

$$\sum_{n_k \in b_J} w_{n,k} \leqslant C_k \;;\; 1 \leqslant k \leqslant d \qquad (4)$$

$$d \leqslant n \qquad (5)$$

We find it a bit different from the bin packing problem. In the multi-capacity bin packing problem, we see number of bins with different capacities and items having varying sizes to fit in every bin. In our problem scenario we have specified number of physical GPP components with max processing capacity, C_T. A list is created consists of current resource usage statistics of VMs, w_i. This list corresponds to the VMs that can be distributed according to constraint (1). This means, resource usage statistics of a migration possible VM needs to be less than, or equal to the available resources in any of the physical component. Capacity C_k of physical GPPs are calculated by measuring remaining processing capacity, that can be derived by deducting current resource usage from threshold capacity C_T. Constraint (2) shows that, for making the migration happen, every destination physical component needs to have capacity to facilitate at least the lowest loaded VM. Constraint (3) means, every virtual machine has to belong to some physical cluster component. Second constraint actually creates the list of d number of GPPs. During migration startup, if all GPPs report their available load levels greater than the lowest loaded VM, then according to constraint (4) number of possible destination hosts and number of VMs will be the same. The goal of this modified VM packing problem is to find a minimal number B of physical GPPs that are capable to serve the VMs. Actual number of VMs and GPPs are the same, i.e. n. Hence, at migration startup according to constraint (5) number of destination physical components d can be same, or less than n. Our goal is to minimize the number of active GPPs facilitating all the VMs. During the model execution VMs are migrated from one machine to another for getting the optimal number of active GPPs by switching off or scaling down the source machine if it is empty.

5.8 Algorithm for VM migration

In this section, we will discuss the algorithm for the model described above. Basically two steps are shown, *squeezing* the cluster size and *releasing* in heavy load situation. We name one of the steps as *releasing* instead of 'expanding' because the number of VMs always remains the same which is equal to the number of physical GPP components. According to our energy-aware system architecture the virtual machine monitor is always aware of processing capacity usage by VM(s) running on it. On the other hand, migration daemon inside cluster front-end keeps track of load statistics of every single GPP component in the cluster. Like state-of-the-art technique of load scheduling, some metrics can be pre-determined to maintain always. This is significant, because for avoiding unnecessary or arbitrary bouncing of VMs among

physical components. Several metrics [KXR+12] can be applied depending on load levels of VMs, or physical components. Considering the ongoing critical user sessions is also necessary. Migration overhead could cause computational overhead in the cluster. In our study we kept this discussion limited, as our goal is to find out possible way of energy reduction. In this optimization algorithm, we define the *highLoad* and *lowLoad* terms like below:

Load level of overall control plane cluster is measured as *lowLoad* according to the pre-defined low load threshold. Migration process triggered if the load level is equal or under *lowLoad*.

In this algorithm, *highLoad* is meant for particular GPP component. It is worth mentioning that in high load situation migration will be triggered in a particular GPP component, if there are multiple VMs running in it. Otherwise that particular GPP will continue operation rejecting any incoming request. The value of *highLoad* is same for every GPP component as all are identical. Migration daemon checks these statistics periodically for every physical component. Hence it knows the situation whether the *highLoad* level is reached, or overall load level goes down to the *lowLoad*. Cluster system is squeezed when the overall load is under, or equal to the *lowLoad* value and released in single *highLoad* situation.

Squeezing situation happens by rearranging VMs to fewer physical components through migration if possible. We depict the *squeezing* state with following steps.

- Migration daemon finds the overall load level of the cluster is equal/under *lowLoad*.
- Virtual machine monitors report to cluster front-end about the processing capacity usage statistics of n VMs by making a list W_i. The list is sorted in increasing order.
- A list containing d number of GPP components is created by cluster front each having C_k capacity according to constraint (1), where, C_T is termed as *highLoad*. The list mentions available capacity in every single physical component to facilitate any virtual machine.
- Every VM W_i is checked starting at the lowest loaded one against every GPP component C_k to check if constraint (3) is satisfied and migrates that VM to the destination GPP. If the constraint is not fulfilled then VM remains in the same GPP as previous.
- Once the migration is done and if no more VM running in the source GPP then it goes to power-save mode, or switched off.

Releasing situation would happen, if *highLoad* limit is reached in any particular GPP

component. VM capacity usage statistics inside that cluster component is achieved through virtual machine monitor and forwarded to migration daemon. Same procedure can be followed like the *squeezing* situation to create the list W_i. Migration daemon also creates the list of GPP according to constraint (1). If constraint (3) is satisfied while checking each VM against every GPP in the list then it is migrated. If the GPP list is empty then new physical component is waken up from inactive mode. As the number of GPPs and number of VMs are the same, so it is obvious that every GPP will contain at least one VM in normal operation or in *highLoad* situation.

5.9 Algorithm implementation

We presume a control plane cluster with ten physical general purpose processor components excluding cluster front-end. In appendix, implementation snippet for the *squeezing* situation is given which can demo virtual machine distribution in low load situation. Result obtained by running the implementation is given in the Figure 5.3 below.

Virtual machine	Current load (%)
1	19
2	57
3	58
4	3
5	44
6	41
7	56
8	79
9	5
10	45

a) Resource usage statistics reported by virtual machine monitor

GPP component	Current load (%)
1 (VM-1)	19
2 (VM-2)	57
3 (VM-3)	58
4 (VM-4)	3
5 (VM-5)	44
6 (VM-6)	41
7 (VM-7)	56
8 (VM-8)	79
9 (VM-9)	5
10 (VM-10)	45

highLoad = 80% of total available capacity of a single cluster component
lowLoad = 40% of total available capacity

b) Current load usage statistics of GPP components obtained from cluster front-end.

GPP cluster component	GPP loads (%)	VM load Statistics inside GPP components
1 (powered down)	0	0
2 (VM-2)	57	57
3 (VM-3)	58	58
3 (powered down)	0	0
5 (VM-5)	44	44
6 (powered down)	0	0
7 (VM-7)	56	56
8 (VM-8)	79	79
9 (VM-9,VM-4,VM-1,VM-6)	68	[5, 3, 19, 41]
10 (VM-10)	45	45

c) Distribution of virtual machines to fewer physical GPP components.

Figure 5.3: Virtual machine migration approach

In this implementation, random loads are considered for triggering the virtual machine migration. Eighty percent of the total GPP load capacity usage is considered as *highLoad*. Also maximum load that a particular VM would handle also considered as the same. While calculating the *aggregated load*, forty percent of the GPP load is considered as *lowLoad*. Virtual machine monitor and GPP cluster front-end would be responsible for operating according to these statistics. Any possible factors (e.g. ongoing user sessions, network overhead due to VM mobility, operational overhead etc.) that might be worth considered (in real implementation) before triggering migration are left

out of consideration for focusing mainly on the optimization model. Implemented codes are thoroughly commented for ease of understanding.

Figure 5.3(a) shows the statistics of virtual machines that are running on General Purpose Processors and corresponding current load (due to traffic operations). Virtual Machine Monitor keeps track of these statistics. The loads shown in the list are generated randomly (taken from 1-80 range) in the code for this demo implementation.

GPP load statistics are shown in figure 5.3(b). As we are studying efficient energy usage through virtual machine migration among GPP cluster, so any particular GPP is able to facilitate more than one virtual machine. In this case, cluster front-end keeps the statistics of GPP loads (not related with VM load). We state these two different loads below for the sake of simplicity:

- VM load – Load of a virtual machine due to traffic operation.
- GPP load – Processing capacity usage due to facilitate virtual machine(s).

Note that, in these two lists (VM and GPP) same load statistics are shown. We presumed an initial situation when every GPP is facilitating one virtual machine each. As every GPP is running one VM, therefore operational load of a particular virtual machine is considered as GPP load. In real implementation, this might be different because, facilitating even a free (no ongoing traffic operation) VM might have some CPU usage of a particular GPP. In this implementation, that situation is kept out of the scope.

In the figure 5.3 (c), sample migration result is depicted. According to the algorithm, virtual machine monitor reports the VM statistics to the migration daemon. Migration daemon gets the load statistics of GPP cluster as well and triggers the migration process. According to our optimization model migration process is triggered sequentially from lower loaded VMs to higher loaded ones. In short, this means lower loaded VMs are migrated first. For example, VM-4 is migrated first, then VM-9 and so forth. First column shows the situation (facilitated VM(s) by a particular GPP) of GPP components inside the cluster after the migration process is completed. Load statistics of GPPs after migration process are shown as percentage. Zero percentage means that, corresponding GPP is powered, or scaled down to save energy consumption. Next column shows corresponding VM loads facilitated by any single GPP component. When migration process is done, VM load statistics are again collected by virtual machine monitor and current/available load statistics of GPP components are gathered by cluster front-end for further migration process.

5.10 Summary

In this chapter, we focused on the possible implementation of hypothetical approach (*squeezing* situation) to reduce energy consumption in control plane cluster (inside a telecommunication infrastructure node) of general purpose processors. The GPP components initially have one virtual machine running on each to perform traffic management activities (e.g. user sessions). In low loaded situation, (within control plane) virtual machines are redistributed to minimal number of GPP components. The rest of the GPP components are then powered down to save energy usage due to the traffic control functionalities. One of the most significant ideas of this approach is to keep the number virtual machines same as the number of host general purpose components. The *squeezing* scenario depicted in this chapter (to impose elasticity through the redistribution of virtual machines in low load situation), is able to reduce 30% of CPU power usage within control plane cluster as we shown that three out of ten processor units are powered down after VM redistribution. Intensive research investigation to measure the more exact percentage of energy reduction is out-scoped in this study as this thesis is primarily targeted to discuss a possible approach to reduce energy consumption.

6 Discussions

The point of interest of this thesis study was traffic management in the control plane (consists of high performance general purpose processing components) inside cluster-based telecommunication infrastructure nodes. We studied one possible way for reducing energy usage in traffic control plane through virtual machine mobility within the cluster. The idea behind this study can be applied to the future telecommunication core network elements especially for traffic operation. There is a steep growth in traffic demand in near future. Hence the interface nodes need to be more energy efficient to achieve green telecommunication activities.

We studied several green factors for core network functionalities that can be optimized to improve energy efficiency in connectivity layer. Several green factors were explored, energy-efficiency-ratios, cooling-requirements, and the energy-efficiency-features. Energy-efficiency-ratios and cooling-requirements were beyond our study scope as we are not dealing with hardware optimization for energy efficiency. But still the second factor can be achieved relatively from this study by energy-efficient control plane operations. The telecommunication infrastructure nodes are multi-rack based complex switching device. The GPP components are placed as blades in racks form the control plane cluster. Control plane energy efficiency would reduce the energy usage per rack. Hence optimized energy usage in racks apparently would produce comparatively less amount of heat in the whole switching device. Therefore, would require less cooling requirements. Discussion of green factors apparently answers the first research question as mentioned in chapter one. And this was the origin of the motivation towards our thesis study. Though this vary first question may lead to several directions of research, but we considered the traffic planes as our goal.

After that our study moved forward to discuss the significance of getting energy aware attitude from traffic operation in telecommunication core network. It is well assumed from this thesis that energy consumption is one of the key factors impacting the operational expenses (OPEX) related to telecommunication infrastructure. Therefore power efficiency in infrastructure nodes and components have immediate impact on the cost of telecommunication providers. We can sketch our reflection like the following Figure 6.1.

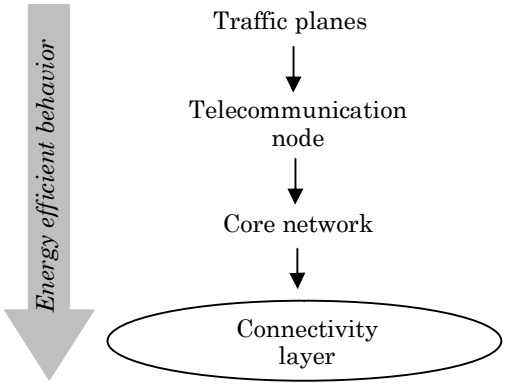

Figure 6.1: Power-aware behavior hierarchy

According to the green factors for telecommunication core network, power-aware behavior of traffic planes can impact the energy efficiency ratios of the equipment. This in turn results into reduced power usage by telecommunication nodes. Therefore we can clearly view the trend in the Figure 6.1 above that apparently converged to energy efficiency in the connectivity layer of horizontally layered network architecture. And we tried to achieve one of the several possible ways of reducing energy consumption due to network operations.

The key driving force of this thesis was to dynamically reduce power usage through software implementation. We studied the system virtualization concept which is the basic of state-of-the-art cloud computing. This thesis did not involve the cloud computing approach but took the virtualization concept. According to our study, system virtualization in every cluster GPP component creates one virtual machine per cluster component. Therefore we get equal number of VMs and physical machines. We studied that the mobility of VMs among the physical GPP components in the control plane cluster could reduce energy usage through elasticity of the active system. VM migration technique could be used for getting mobility behavior. We discussed and studied the pre-copy migration approach as the possible migration technique in control plane cluster. Elastic behavior works by *squeezing* and *releasing* the system. In low traffic situation VMs could be redistributed among fewer number of physical components to keep rest of the GPPs switched off, or scaled down. On the other hand, during high traffic situation in any particular physical component extra (if the particular GPP contains more than one virtual machine) VM(s) would go back to another, or newly waken up component. As all physical components are identical, hence VM mobility could be implemented without considering underlying hardware.

The energy consumption of network is a major issue in the future network technologies

especially in telecommunication sectors due to excessive demand of traffic-grabber services and applications. This thesis showed a possible way of energy efficiency in the core telecommunication infrastructure node of horizontally layered network architecture. Study focused mainly on traffic control plane of cluster-based infrastructure node that may decrease the energy consumption to a great extent. In following sections, we discuss about several other techniques that can be explored to gain more energy reduction through power aware system implementation for handling the traffic.

6.1 Possible alternatives to optimize energy usage

Energy efficiency in telecommunication core network is always a burning issue both at present and in future telecommunication network technologies. The amount of power usage due to telecommunication network activities and core network traffic manipulation is increasing very rapidly. In horizontal network architecture, telecommunication core network is a part of connectivity layer where traffic load due to end user data connections is a critical point to deal with. Significant amount of energy is consumed due to traffic handling and processing. Here the trade-off is, we can never compromise the network traffic situation with system optimization but still we need to find any holistic approach to minimize energy consumption. From the hardware point of view, devices and processing components can always be designed and manufactured to operate with less energy. But researchers also need to make efforts to reduce energy usage dynamically through new system implementation for network traffic handling and processing activities. However the trade-off between energy-usage and performance requirements needs to be considered always while tailoring the system to achieve energy optimization. In this chapter, discussion will go forward about several further possibilities that might help to reduce energy usage in traffic operations inside telecommunication infrastructure nodes.

6.1.1 Distribution of control functionalities

In the current telecommunication or mobile telephony, switches separate the implementations of control plane functionalities from data plane traffic functions (packet forwarding and processing). As we see in this sort of architecture control plane functions are implemented in GPP components resides in a cluster. Every cluster component connected to several network processors of data plane. In future, telecommunication nodes can be designed to implement control plane functionality to

networked processors of data plane. This actually means control plane functions can be distributed to data plane along with control elements for increased scalability. The challenge in this sort of implementation can be, determining the control functions that can be distributed and the data plane elements that can host these functions. The possible hypothetical architecture is shown in the Figure 6.2 below with simple diagram.

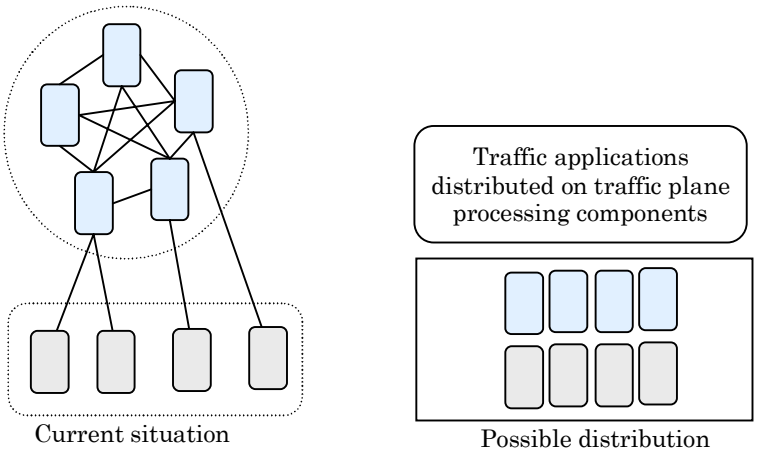

Figure 6.2: Distribution of traffic functionalities

If the distribution can be done to data plane processing components than the resiliency and scalability issues may be significantly reduced due to distribution. Challenge may arise for computational overhead to determine distribution candidate but that can be also mitigate through comprehensive research. Proper work-load balancing and functional distribution will apparently ensure more optimal energy usage of entire infrastructure node.

6.1.2 Cloud-based traffic operation

In a mobile telephony switch, control plane and data plane are two traffic planes. These traffic planes are actually doing traffic handling and traffic processing. Every GPP in the control plane is tightly connected with one or more application specific processor(s). The programmability and performance are totally different of a control and forwarding element. Therefore researchers can give efforts to logically separate their functionality. If the separation is deployed, then control plane elements can always remain connected with any data plane element. Aggressive network, application or energy optimization capability can be developed in separated architecture. We show the Figure 6.3 below for the possible separation from functional point of view.

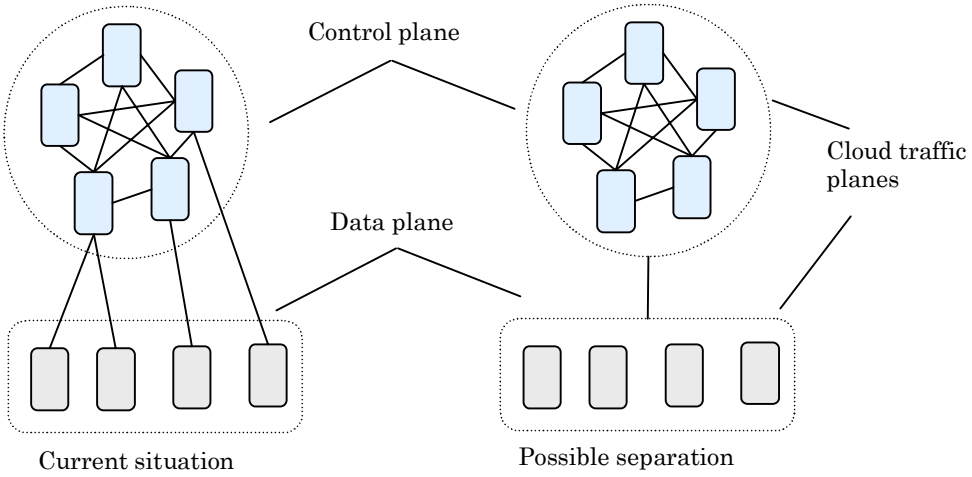

Figure 6.3: Separating control and data plane with different cloud deployment

All the data plane network processors and control plane general purpose processors can be resided in a cloud. Appropriate implementation of virtualization layer can be imagined over all the physical traffic plane components. Any kind of user session request coming in will be served from the virtualization layer by the cloud of control functions. After data session establishment, for traffic processing system will communicate with virtualized forwarding element which may serve the computation and actual data movement. The communication between control plane cloud and data plane cloud can be investigated through intensive research.

6.1.3 Additional explorations

In addition to two significant possibilities of future research studies (mentioned in sections 6.1.2 and 6.1.2) several more axes can be depicted briefly that can be elevated from this thesis. We sketch the Figure 6.4 below to point out possible further studies.

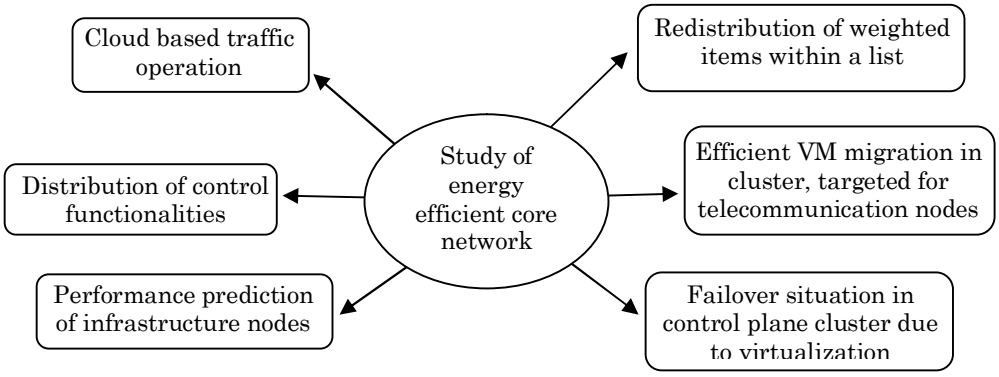

Figure 6.4: Possible study exploration axes

In this thesis for rearranging virtual machine we mentioned multiple-bin-packing

problem. The problem scenario of VM distribution among fewer physical machines has a list of GPP components with different amount of free resources. Whereas in bin-packing problem, multiple empty bins with different capacities are item destinations. More research efforts can be given to gain efficient rearrangement. Efficiency can be measured by low migration time, less computational overhead, less impact on internal network bandwidth, or lower service downtime due to migration.

Our study did not cover the failure situation handling in control plane cluster. Instead we studied only the redundancy of cluster front-end. Failure situation can be observed at any particular time which means any single physical GPP component can be crashed or malfunctioning. This can be another axe that is open for study further.

This thesis illustrated pre-copy virtual machine migration in control plane cluster. Live VM migration is an intensive area of research in system virtualization area. Efficiency of successful migration depends on several metrics like total-migration-time, down-time, ongoing application interruption, writable working set, network overhead, or migration overhead. Especially for telecommunication infrastructure node these issues can be explored in details to trace out the efficient way that is appropriate for mobile core network infrastructure components. While applying multiple methods of VM migration, the performance prediction is also significant to investigate. Prediction can involve better performance in traffic-control such as user session establishment rate. Optimal performance is always necessary because of the trade-off between the node performance and energy reduction. All possible studies mentioned here are opt for gaining energy efficiency by means of software system implementation. Energy efficient hardware design may also reduce power usage but, in this thesis we out scoped hardware related issues. The idea is that, whatever hardware is used system can be optimized always.

6.2 Summary

In this thesis, we discussed about the possible approach that can be applied in control plane traffic operations for gaining energy efficient behavior. We discussed power aware architecture of the system which was significant, because for every possible system implementation it is always crucial to give an overall working concept of the system. The most worth mentioning point here is that, this system is not targeted for any specific traffic situation of the overall network like peak, or off-peak hours. If implemented it could show power aware behavior online with the varying traffic load situation. We studied that for avoiding redundant computational overhead it would be

better to maintain a clock system for performing possible migration. Study discussed that, this is necessary because at a certain time, control plane cluster may not have any request to establish user session but still have traffic operation ongoing in lower level ASPs (Application Specific Processor). Though our possible migration approach does not show any noticeable interruption in operations running inside VM but, it is better to wait for specific time period to check for migration. This waiting time could be pre-determined according to SLA to avoid extra computational overhead due to migration. Several measurement metrics such as the *total migration*, *down time* and also computational overhead might have impact on operational performance of traffic handling in the cluster. These are also noteworthy because migration of a particular virtual machine from source to destination is not trivial while minimizing the noticeable interruption of services running inside. Even though the control plane cluster is a high performance cluster, but still the implementation of VM mobility needs to be network-aware [PiY10], application aware [SZL+11] and also at the same time needs to minimize the *total migration time*. All of these metrics may greatly impact the energy efficient behavior of the control plane that can be achieved through elasticity of the GPP cluster.

List of references

[ASR+10] Akoush, S., Sohan, R., Rice, A., Moore, A.W., Hopper, A., Predicting the Performance of Virtual Machine Migration. *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, Miami Beach, Florida, USA, Aug. 17-19, 2010 pages 37-46.

[BBB07] Bakmaz, B., BojkoviC, Z., Bakmaz, M., Internet Protocol Multimedia Subsystem for Mobile Services. *Systems, Signals and Image Processing, 2007 and 6th EURASIP Conference focused on Speech and Image Processing, Multimedia Communications and Services. 14th International Workshop on*, 27-30 June 2007, pp.339,342

[BLW99] Bsrje, J., Lund, H-, Wirkestrand, A., Real-time routers for wireless networks. *Ericsson Review*, Vol. 76(1999): 4, pages 190-197.

[Bos10] Bose, B.K., Global Warming: Energy, Environmental Pollution, and the Impact of Power Electronics. *Industrial Electronics Magazine, IEEE*, March 2010, vol.4, no.1, pp.6,17.

[CFH+05] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., Warfield, A., Live migration of virtual machines. *2nd Symposium on Networked Systems Design and Implementation (NSDI'05),* Boston, Massachusetts, USA, May 2-4, 2005, pages 273–286.

[CGa04] Camarillo, G., García-Martín M-A., The 3G IP Multimedia Subsystem (IMS). *Merging the Internet and the Cellular Worlds, Chichester*, West Sussex, UK, Wiley, 2004.

[CLN10] Chaisiri, S., Lee, B.-S., Niyato, D., Robust cloud resource provisioning for cloud computing environments. *Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on*, Perth, Australia, Dec. 13-15, 2010, pages 1-8.

[DRB03] Drempetic, T., Rihtarec, T., Bakic, D., Next generation networks architecture for multimedia applications. *Video/Image Processing and Multimedia Communications, 2003. 4th EURASIP Conference focused on*, July 2003, vol.2, pp.563,568.

[DYS+10] Du, Y., Yu, H., Shi, G., Chen, J., Zheng, W., Microwiper: Efficient Memory Propagation in Live Migration of Virtual Machines. *Parallel Processing (ICPP), 2010 39th International Conference on*, San Diego, California, USA, Sept. 13-16, 2010, pages 141-149.

[FWB07] Fan, X., Weber, W.-D., Barroso, L. A., Power provisioning for a warehouse-sized computer. In *ISCA '07: Proc. of the 34th annual international symposium on Computer architecture*, New York, NY, USA, Jun. 2007, pages 13–23.

[HCZ10] Huan, Y., Changyun, M., Zhigang, W., The design of IP telephony media gateway which is based on soft-switching technology. *Computer Design and Applications (ICCDA), 2010 International Conference on*, Qinhuangdao, Hebei, China, Jun. 25-27, 2010, pages V5-379 - V5-381.

[HGW+11] Huang, Q., Gao, F., Wang, R., Qi, Z., Power Consumption of Virtual Machine Live Migration in Clouds. *Communications and Mobile Computing (CMC), 2011 Third International Conference on*, Qingdao, China, Apr. 18-20, 2011, pages 122-125.

[HiG09] Hines, M. R., Gopalan, K., Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proc. of the ACM/Usenix International Conference on Virtual Execution Environments (VEE'09)*, Washington DC, Mar. 2009, pages 51-60.

[HJL+08] Hu, L., Jin, H., Liao, X., Xiong, X., Liu, H., Magnet: A novel scheduling policy for power reduction in cluster with virtual machines. In *Proc. of the IEEE International Conference on Cluster Computing (Cluster'08)*, Tsukuba, Japan, Sep. 29 – Oct. 1, 2008, pages 13–22.

[HLA+06] Huang, W., Liu, J., Abali, B., Panda, D. K., A case for high performance computing with virtual machines. In *Proc. of the 20th ACM International Conference on Supercomputing (ICS'06)*, Cairns International Hotel, Queensland Australia, Jun. 28 – Jul. 01, 2006, pages 125-134.

[HLM+09] Hermenier, F., Lorca, X., Menaud, J.-M., Muller, G., Lawall, J., Entropy: a consolidation manager for clusters. In *Proc. of the ACM/Usenix International Conference on Virtual Execution Environments (VEE'09)*, Washington DC, Mar. 2009, pages 41–50.

[JDW+09] Jin, H., Deng, Li., Wu, S., Shi, X., Pan, X., Live virtual machine migration with adaptive, memory compression. *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, New Orleans, Louisiana, USA, Aug. 31 - Sept. 4, 2009, pages 1-10.

[Kha12] Kharitonov, D., Green telecom metrics in perspective. *Communications (APCC), 2012. 18th Asia-Pacific Conference on*, Oct. 15-17, 2012, pp.548, 553.

[KSK+12] Kuwashima, T., Sekimoto, K., Kawai, K., Iida, Y., Yokoyama, R., Takemoto, M., Fujioka, Y., Yoshida, Y., Neutralize CO2 emissions by Product Contributions. Electronics Goes Green 2012+ (EGG), September 2012, pp.1,5, 9-12.

[KXR+12] Kejiang Ye, Xiaohong Jiang, Ran Ma, Fengxi Yan, VC-Migration: Live Migration of Virtual Clusters in the Cloud. *Grid Computing (GRID), 2012. ACM/IEEE 13th International Conference on*, Sept. 20-23, 2012, pp.209-218.

[LeJ03] Lee, B.K., John, L.K., NpBench: a benchmark suite for control plane and data plane applications for network processors. *Computer Design, 2003. Proceedings. 21st International Conference on*, San Jose, California, Oct. 13-15, 2003, pages 226- 233.

[LHL+08] Li, Q., Huai, J., Li, J., Wo, T., Wen, M.; HyperMIP: Hypervisor Controlled Mobile IP for Virtual Machine Live Migration across Networks. *High Assurance Systems Engineering, 2008 11th IEEE International Syposium on*, Nanjing, China, Dec. 3-5 2008, pages 80-88.

[LJL+09] Liu, H., Jin, H., Liao, X., Hu, L., Yu, C., Live migration of virtual machine based on full system trace and replay. In *Proc. of the 18th International Symposium on High Performance Distributed Computing (HPDC'09),* Garching near Munich, Germany, Jun. 11-13, 2009*,* pages 101–110.

[LJL+11] Liu, H., Jin, H., Liao, X., Yu, C., Xu, C., Live Virtual Machine Migration via Asynchronous Replication and State Synchronization. *Parallel and Distributed Systems, IEEE Transactions on*, volume: 22, issue: 12 (Mar. 2011), pages 1986-1999.

[LKK99] Leinberger, W., Karypis, G., Kumar, V., Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *proc. of 1999 International Conference on Parallel Processing*, Aizu-Wakamatsu City, Japan, Sept. 21-24, 1999, pages 404-412.

[MLL10] Ma, F., Liu, F., Liu, Z., Live virtual machine migration based on improved pre-copy approach. *Software Engineering and Service Sciences (ICSESS), 2010*

IEEE International Conference on, Beijing, China, Jul. 16-18, 2010, pages 230-233.

[MoC10] Moghaddam, F.F., Cheriet, M., Decreasing live virtual machine migration down-time using a memory page selection based on memory change PDF. *Networking, Sensing and Control (ICNSC), 2010 International Conference on*, Chicago, IL, USA, Apr. 10-13, 2010, pages 355-359.

[NaM07] Nam, D.-h., Min, H.-k., An Energy-Efficient Clustering Using a Round-Robin Method in a Wireless Sensor Network. In *Proc. of 5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA2007)*, Aug. 20-22, 2007, pages 54-60.

[NaS07] Nathuji, R., Schwan, K., Virtual power: Coordinated power management in virtualized enterprise systems. In *Proc. of the 21st ACM Symposium on Operating Systems Principles (SOSP'07)*, Stevenson, WA, USA, Oct. 2007, pages 265–278.

[PKC+09] Park, J.-G., Kim, J.-M., Choi, H., Woo, Y.-C., Virtual machine migration in self-managing virtualized server environments. *Advanced Communication Technology, 2009, ICACT 2009, 11th International Conference on,* Phoenix Park, Korea, Feb. 15-18, 2009, vol.03, pages 2077-2083.

[PiY10] Piao, J.T., Yan, J., A Network-aware Virtual Machine Placement and Migration Approach in Cloud Computing. *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, Nanjing, Jiangsu, China, Nov. 1-5, 2010, pages 87-92.

[OSS+02] Osman, S., Subhraveti, D., Su, G., Nieh, J., The design and implementation of zap: A system for migrating computing environments. In *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, Dec. 9-11, 2002, pages. 361–376.

[SKV+12] Shicong Meng, Kashyap, S.R., Venkatramani, C., Ling Liu, Resource-Aware Application State Monitoring. *Parallel and Distributed Systems, IEEE Transactions on*, vol.23, no.12, (Dec. 2012) pp.2315, 2329.

[SZL+11] Shrivastava, V., Zerfos, P., Lee, K., Jamjoom, H., Yew-Huey, Liu, Banerjee, S., Application-aware virtual machine migration in data centers. INFOCOM, 2011 Proceedings IEEE, Shanghai, China, Apr. 10-15, 2011, pages 66-70.

[SCP+02] Sapuntzakis, C. P., Chandra, R., Pfaff, B., Chow, J., Lam, M. S., Rosenblum, M., Optimizing the migration of virtual computers. In *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, Dec. 9-11, 2002, pages 377–390.

[TJL+08] Tianzhou C., Jiangwei, H., Liangxiang, X., Zhenwei, Z., A Practical Dynamic Frequency Scaling Scheduling Algorithm for General Purpose Embedded Operating System. *Future Generation Communication and Networking, 2008. FGCN '08. Second International Conference on*, Dec. 13-15, 2008, pages 213-216.

[WCS+04] Whitaker, A., Cox, R. S., Shaw, M., Grible, S. D., Constructing services with interposable virtual hardware. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI'04)*, San Francisco, California, USA, Mar. 29-31, 2004, pages 169-182.

[VNE+08] Vallee, G., Naughton, T., Engelmann, C., Ong, H., Scott, S.L., System-Level Virtualization for High Performance Computing. In *Proc. of the 16th Euromicro Conference on Parallel, Distributed and Network-based Processing*, Toulouse, France, Feb. 13-15, 2008, pages 636-643.

[WCS+04] Whitaker A.; Cox R. S.; Shaw M.; Gribble S. D., Constructing services with interposable virtual hardware. In *Proc. of the 1st USENIX Symposium on*

Networked Systems Design and Implementation (NSDI'04), San Francisco, California, Mar. 29-31, 2004, pages 169–182.

[Wit00] Witzel, A., Control servers in core network. *Ericsson Review,* Vol. 77(2000): 4, pages 234-243.

[WMS11] Wilcox, D., McNabb, A., Seppi, K., Solving virtual machine packing with a Reordering Grouping Genetic Algorithm. *Evolutionary Computation (CEC), 2011 IEEE Congress on*, New Orleans, LA, USA, Jun. 5-8, 2011, pages 362-369.

[XHG+06] Xiaolong, Q., He, S., Guo, D., Jing, Y., On Time-Critical Data Transmission of EtherNet/IP. *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, Dalian, China, Jun. 21-23, 2006, pages 4623-4625.

[Zay87] Zayas, E., Attacking the process migration bottleneck. In *Proc. of the Eleventh ACM Symposium on Operating System Principles*, Austin, Texas, Nov. 8-11, 1987, pages 13–24.

Appendix - Virtual machine distribution algorithm

One significant part of this thesis was to write code for obtaining the result of virtual machine distribution algorithm (mock implementation). In this implementation, *squeezing* scenario within GPP cluster components (by migrating virtual machines) in low load situation is depicted. Possible VM distribution result shown in the section 5.8 is taken by running this code block. The code is written in JAVA. Random load statistics (for both GPPs and VMs) are taken into consideration while implementing this code. The motivation behind this implementation was to show, how the VM mobility could squeeze the GPP cluster.

A Fields and methods summary

Predefined values

MAX_LOAD_LEVEL: This is the maximum load level which is meant for the processing capacity of any particular GPP in the cluster. Eighty percent is considered as maximum.

LOW_LOAD_LEVEL: This is the pre-determined threshold *lowLoad* level which is compared with the overall aggregated load of the cluster for taking migration decision.

NUMBER_OF_VM: Total number of GPP except the cluster front-end is presumed to ten. Hence according to our study, total number of VM is also ten.

Fields used

staticGPPLoadStats: At the beginning of migration, this map contains overall load situation of all the GPPs in cluster. Facilitator GPP(s) is (are) determined at the migration startup meaning that, if any particular GPP has available load capacity to facilitate additional virtual machine, or not. This list contains all active and inactive (powered down) GPPs while storing 'Zero' load as powered down indication.

vmLocationList: This list is provided by virtual machine monitor that contains virtual machine locations. As we have constant number (i.e. ten) of VMs running inside the GPP components, so this list preserves the information about "which VM is running on what number of GPP".

overallGPPLoadStats: This Map contains the comprehensive list of GPP components that shows the set of VMs running on a particular GPP component. This list only

contains the GPP that are in active mode meaning that, running at least one VM inside a particular GPP component.

*VMLoadStats***:** Load usage of every single virtual machine is stored in this Map. This apparently helps in taking migration decision. If there is any heavily loaded VM, or any one running very critical user sessions will not be migrated. In this implementation we only considered heavily loaded situation.

Methods

*getInitialVMStatistics()***:** For getting the load statistics and location list from virtual machine monitor this method is used to initialize vmLocationList and VMLoadStats

migrationDecision(Map<Integer, Integer> , Map<Integer, Integer>)

This method is responsible for taking migration decision after comparing the cluster's load situations. Two different checks are performed:

- Checks the aggregated load level of the cluster is under *lowLoad* threshold. This triggers squeeze operation if succeeds.
- Next check is performed to investigate the load level of any particular GPP component. One helper method *isHighLoadLimitExceeded(Map<Integer, Integer>)* is invoked which returns a Boolean value depending on whether the load situation of any particular GPP component is reached to *highLoad* or not. If so then it triggers the release operation.
- In case both of the checks fail then cluster operation continues as usual.

squeezeCluster (Map<Integer, Integer>, Map<Integer, Integer>)

This method is responsible for migration, or virtual machine distribution in low load situation. This requires, VMLoadStats, staticGPPLoadStats as parameters. Several methods are called from inside this:

- *availableLoadList(Map<Integer, Integer>):* This is intended to create an intermediate list of GPPs with their corresponding available loads.
- *Map sortByLoad(Map, boolean):* According to our studied algorithm this method sorts the lists of GPPs and VMs for impacting the migration decision.

releaseCluster(Map<Integer, Integer>, Map<Integer, List>, Map<Integer, Integer>))

This method is not implemented in this demo as we tried to depict only the squeeze situation.

B Code Implementation

```
package control.plane.vm.migration;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Random;

@SuppressWarnings({ "rawtypes", "unchecked", "unused" })
public class MigrationOperation {

/** High Load threshold */
    public static final int MAX_LOAD_LEVEL = 80;

/** Low Load label */
    private static final int LOW_LOAD_LEVEL = 40;

/**
 * According to mentioned algorithm number of GPP and VM are the same It is
 * presumed here that, total 10 GPPs are in the GPP-Cluster, hence total
 * number of VM is also 10
 */
    public static final int NUMBER_OF_VM = 10;
    public static final int ZERO_LOAD = 0;

/**
 * This contains information about which VM resides in what numbered GPP
 * among the cluster
 */
    public static int[] vmLocationList = new int[11];

/** Always hold the statistics of all GPPs with their current load usage */
    public static Map<Integer, Integer> staticGPPLoadStats = new
HashMap<Integer, Integer>();

/**
 * Container map of GPP(s), who have VM(s) running in it. This map is
 * updated during migration and populates post migration statistics to
 * 'staticGPPLoadStats'
 */
    public static Map<Integer, List> overallGPPLoadStats = new
HashMap<Integer, List>();

/**
 * Static container of all VMs (number of VM = number of GPP) with their
 * current load usages
 */
    public static Map<Integer, Integer> VMLoadStats = new HashMap<Integer,
Integer>();
```

```
private static Object sumOfLoad;

    public static void main(String args[]) {

        getInitialVMStatistics();
        staticGPPLoadStats = getCurrentGPPLoadStatistics();

        migrationDecision(VMLoadStats, staticGPPLoadStats);

        System.out.format("VM No.  | Load Usage (Unit)\n---------------\n");
        showLoadStatistics(VMLoadStats);
        System.out.format("GPP No. | Current Load (Unit)\n-------------\n");
        showLoadStatistics(staticGPPLoadStats);
        System.out.format("GPP No. | Loaded VMs\n----------------------\n");
        showLoadStatistics(overallGPPLoadStats);
    }

    private static void migrationDecision(Map<Integer, Integer> vmLoadStats,
                   Map<Integer, Integer> staticGPPLoadStats) {

             if (getLoadLevel(staticGPPLoadStats) < getLowLoadLabel()) {
                   squeezeCluster(vmLoadStats, staticGPPLoadStats);
             }
             else if(isHighLoadLimitExceeded(staticGPPLoadStats)) {
                   releaseCluster(vmLoadStats, overallGPPLoadStats,
staticGPPLoadStats);
             }
             else {
                   /** Continue normal operation */
                   return;
             }
       }

       private static int getLoadLevel(Map<Integer, Integer>
staticGPPLoadStats) {
             double sumOfLoad = 0;
             Iterator iterator = staticGPPLoadStats.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry pairs = (Map.Entry) iterator.next();
            sumOfLoad += ((Integer) pairs.getValue()).doubleValue();
        }
             return (int)Math.ceil(sumOfLoad/staticGPPLoadStats.size());
       }

       private static void squeezeCluster(Map<Integer, Integer> vMLoadStats,
            Map<Integer, Integer> gppLoadStats) {

        Map<Integer, Integer> gppAvailableLoad = sortByLoad(
                availableLoadList(gppLoadStats), true);
        Map<Integer, Integer> vmLoads = sortByLoad(vMLoadStats, false);

        Iterator itVM = vmLoads.entrySet().iterator();
        while (itVM.hasNext()) {
            Map.Entry vmPairs = (Map.Entry) itVM.next();

            Iterator itGPP = gppAvailableLoad.entrySet().iterator();
            Iterator it = overallGPPLoadStats.entrySet().iterator();
```

```
            int vmLocation = vmLocationList[((Integer) vmPairs.getKey())
                    .intValue()];

            Integer destGPPindex = new Integer(vmLocation);

/**
 * Skips the GPP that is already in Sleep/Scaled-down mode Or, if
 * the GPP that has already a list of VMs running in it
 */
            if (overallGPPLoadStats.get(destGPPindex) == null
                    || overallGPPLoadStats.get(destGPPindex).size() > 1)
                continue;

            while (itGPP.hasNext()) {
                Map.Entry gppPairs = (Map.Entry) itGPP.next();

                int gppIndexValue = ((Integer)
gppPairs.getKey()).intValue();
                if (vmLocation == gppIndexValue)
                    continue;

                if (checkAddVMLoad(gppPairs.getValue(), vmPairs.getValue()))
{
                    List<Integer> loadList = new ArrayList<Integer>();
                    loadList = overallGPPLoadStats.get(gppPairs.getKey());
                    loadList.add((Integer) vmPairs.getValue());
                    overallGPPLoadStats.put((Integer) gppPairs.getKey(),
                            loadList);

                    gppPairs.setValue(availableLoad(sumList(loadList)));

                    Integer gppIndex = new Integer(
                            vmLocationList[((Integer) vmPairs.getKey())
                                    .intValue()]);
/**
 * Condition below removes corresponding ITEM from
 * Facilitator-GPP map (lists available capacities) who does
 * not contain anymore VM, so the same indexed GPP can be
 * SCALED DOWN Facilitator-GPP map only contains those, who
 * have VM running in it and also has capacity available for
 * facilitating additional VM Facilitator-GPP map is an
 * intermediate list of GPP(s) used for migration operation.
 */
                    if (((Integer)
sumList(overallGPPLoadStats.get(gppIndex)))
                            .intValue()
                            - ((Integer) vmPairs.getValue()).intValue() ==
0) {
                        gppAvailableLoad.remove(gppIndex);
                    }

                    vmLocationList[((Integer) vmPairs.getKey()).intValue()]
= ((Integer) gppPairs
                            .getKey()).intValue();

                    /**
                     * Scales down GPP in overallGPPLoadStats map.
                     */
```

```
                    overallGPPLoadStats.remove(vmLocation);
                    break;
                }
            }
        }
/**
 * After migration completion final operational statistics is populated
 * to static 'staticGPPLoadStats' map (updates the current load usage)
 * Assigns 'ZERO_LOAD' to corresponding GPP(s) whose VM(s) are migrated
 * away meaning that, it went to Scaled-Down/Sleep state
 */
        Iterator it = staticGPPLoadStats.entrySet().iterator();
        while (it.hasNext()) {
            Map.Entry pairs = (Map.Entry) it.next();
            if (overallGPPLoadStats.get(pairs.getKey()) == null) {
                pairs.setValue(new Integer(ZERO_LOAD));
                continue;
            }

pairs.setValue(sumList(overallGPPLoadStats.get(pairs.getKey())));
        }
        // int i= 0;
    }


      private static boolean isHighLoadLimitExceeded(Map<Integer, Integer>
staticGPPLoadStats) {
             Iterator iterator = staticGPPLoadStats.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry pairs = (Map.Entry) iterator.next();
            if(((Integer)pairs.getValue()).intValue() >= MAX_LOAD_LEVEL) {
             return true;
            }
        }
             return false;
      }

      private static void releaseCluster(Map<Integer, Integer> vMLoadStats,
            Map<Integer, List> overallGPPLoadStats, Map<Integer, Integer>
gppLoadStats) {
             /** TODO: Functional block to Power up inactive VMs among
distribute VMs among them */
      }

    private static Object sumList(List<Integer> loadList) {
        Integer sum = 0;
        for (Integer i : loadList)
            sum = sum + i;
        return sum;
    }

/**
 * Initially we presume every GPP is serving at least one VM So at the
 * beginning we get the same list for GPPLoadList and VMLoadList
 *
 * @return
 */
    private static Map<Integer, Integer> getCurrentGPPLoadStatistics() {
```

```
        Iterator iterator = null;

/**
 * If any GPP currently facilitating at least one VM then, calculate
 * available capacity accordingly
 */
        if (staticGPPLoadStats.size() == 0)
            iterator = VMLoadStats.entrySet().iterator();
        else {
            updateOverallGPPStats(staticGPPLoadStats);
            return staticGPPLoadStats;
        }

        while (iterator.hasNext()) {
            Map.Entry pairs = (Map.Entry) iterator.next();
            staticGPPLoadStats.put(new Integer(pairs.getKey().toString()),
                    new Integer(pairs.getValue().toString()));
        }
        updateOverallGPPStats(staticGPPLoadStats);
        return staticGPPLoadStats;
    }

    public static int getLowLoadLabel() {
       return LOW_LOAD_LEVEL;
    }

       private static void updateOverallGPPStats(
            Map<Integer, Integer> currentGPPLoadStats) {

        Iterator it = overallGPPLoadStats.entrySet().iterator();

        Iterator iterator = currentGPPLoadStats.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry pairs = (Map.Entry) iterator.next();

/**
 * If there is no load on any certain GPP, then operational GPP map
 * disregards that from facilitator list, because that GPP is
 * already in sleep/scaled-down mode.
 */
            if (((Integer) pairs.getValue()).intValue() == ZERO_LOAD)
                continue;

            List<Integer> vm = new ArrayList<Integer>();
            if (it.hasNext()) {
                Map.Entry entry = (Map.Entry) it.next();
                vm = (List<Integer>) entry.getValue();
            }
            vm.add(new Integer(pairs.getValue().toString()));
            overallGPPLoadStats.put(new Integer(pairs.getKey().toString()),
vm);
        }
    }

/**
 * This method is intended to get the load statistics of VMs from Virtual
 * machine manager In this demo implementation this just returns random load
```

```
 * statistics
 */
    public static void getInitialVMStatistics() {

        int load, index;
        for (index = 0; index < NUMBER_OF_VM; ++index) {
            load = (new Random()).nextInt(MAX_LOAD_LEVEL);
            VMLoadStats.put(new Integer(index + 1), new Integer(load));
            vmLocationList[index + 1] = index + 1;
        }
    }

    public static void showGPPVMLoads(int[] GPPLoadStats) {

        for (int index = 0; index < 5; ++index) {
            System.out.println("GPP No.: " + (index + 1) + " "
                    + GPPLoadStats[index]);
        }
    }

    public static void showLoadStatistics(Map gppMap) {
        Iterator iterator = gppMap.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry pairs = (Map.Entry) iterator.next();
            /*
             * if(pairs.getValue() instanceof List) { continue; }
             */
            System.out.format("%4s    |  %s\n", pairs.getKey().toString(),
                    pairs.getValue().toString());
            System.out.format("----------------------------\n");
            iterator.remove();
        }
        System.out.format("\n");
    }

    public static boolean checkAddVMLoad(Object availableGPPLoad,
            Object extraVMLoad) {
        return Integer.parseInt(availableGPPLoad.toString()) >= Integer
                .parseInt(extraVMLoad.toString());
    }

/**
 * Computes remaining capacity of a particular GPP
 * @param currentGPPLoad
 * @return
 */
    public static int availableLoad(Object currentGPPLoad) {
        return MAX_LOAD_LEVEL - Integer.parseInt(currentGPPLoad.toString());
    }

    public static Map<Integer, Integer> availableLoadList(
            Map<Integer, Integer> gppMap) {

        Map<Integer, Integer> loadMap = new HashMap<Integer, Integer>();
        loadMap.putAll(gppMap);

        Iterator iterator = loadMap.entrySet().iterator();
        while (iterator.hasNext()) {
```

```
            Map.Entry pairs = (Map.Entry) iterator.next();

/**
 * Skips the Scaled-down/Slept GPP from the VM-facilitator GPP list
 */
            if (((Integer) pairs.getValue()).intValue() == ZERO_LOAD) {
                iterator.remove();
                continue;
            }
            pairs.setValue(availableLoad(pairs.getValue()));
        }
        return loadMap;
    }

    private static Map sortByLoad(Map map, final boolean descending) {

        Map loadList = new HashMap<Integer, Integer>();
        loadList.putAll(map);

        List list = new LinkedList(loadList.entrySet());

        Collections.sort(list, new Comparator() {
            public int compare(Object o1, Object o2) {
                if (descending)
                    return ((Comparable) ((Map.Entry) (o2)).getValue())
                            .compareTo(((Map.Entry) (o1)).getValue());

                return ((Comparable) ((Map.Entry) (o1)).getValue())
                        .compareTo(((Map.Entry) (o2)).getValue());
            }
        });

/**
 * put sorted list into map again LinkedHashMap make sure order in which
 * keys were inserted
 */
        Map<Integer, Integer> sortedLoadList = new LinkedHashMap<Integer,
Integer>();
        for (Iterator it = list.iterator(); it.hasNext();) {
            Map.Entry entry = (Map.Entry) it.next();
            sortedLoadList.put(new Integer(entry.getKey().toString()),
                    new Integer(entry.getValue().toString()));
        }
        return sortedLoadList;
    }
}
```