Energy Efficient Rescheduling Algorithm for High Performance Computing

Manisha Chauhan[1], Nazia Parveen[2], Sumit Kumar Saurav[3], Ganga Prasad G.L[4]
Centre for Development of Advanced Computing, 'C-DAC Knowledge Park',
Opp.HAL Aeroengine Division, No.1, Old Madras Road, Byappanahalli,
Bengaluru-560038, Karnataka, INDIA
manishac@cdac.in[1],naziap@cdac.in[2],sumitk@cdac.in[3],gpr@cdac.in[4]

Abstract **– With the augmentation of High Performance Computing (HPC) system and power consumption pattern, energy optimization has become an important concern. Several surveys indicate that the energy utilized in computation and communication within a HPC system contributes considerably to their operational costs. The proposed energy efficient rescheduling algorithm aims to reduce energy consumption in HPC. This algorithm is based on energy efficient rescheduling of a job which considers the optimal operating point (OOP) i.e. voltage and frequency along with resource matching constraint to achieve the target performance. Performance –Energy-Time (PET) matrix for every application is built and stored in knowledge base (KB) in order to devise its OOP. The devised OOP gives minimal energy consumption at target execution environment. The algorithm is useful for energy optimization for cluster environment. Earlier works exploited process migration technique to address load balancing aspect. The proposed algorithm exploits process migration and reschedules the job dynamically with its OOP in context of energy reduction.**

Keywords—High Performance Computing, Energy Optimization, Process Migration, Knowledge Base, Energy Efficient Rescheduling Algorithm .

I. INTRODUCTION

Power Management in HPC infrastructure has become a critical concern. The operating cost, reliability and environmental impact are the prime concern in operation of HPC systems [1]. In advent of multi core technology, advancement in chip miniaturization and the component based hardware abstraction has pave the way for researchers to exploit these abstraction to have better control on execution environment. The power dissipation leads to heat generation and results show that with every 10 degree Celsius increase in temperature, the probability of system failure gets doubled [2]. Ryan et al [3] states that the energy efficient computing has become a necessity in context of computation as power is a limiting factor.

The devised algorithm is based on application Performance-Energy-Time (PET) matrix. PET matrix is an analysis of application in terms of performance, energy consumption and execution time to determine its OOP. The algorithm reschedules the application at run time to save energy. For energy optimization, firstly we measured the energy consumption of various workloads at different voltage and frequency levels to identify OOP and then stored the details of the architecture of the system, CPU voltage and frequency level, number of nodes used, cores used per node, energy consumption, application execution time and problem size etc. in KB. While scheduling or rescheduling the job on compute nodes, we are considering the application OOP apart from the resource availability and system utilization.

A. Scope and Objectives

The objective of the work presented in this paper is to minimize energy consumption and maximize system utilization. The proposed algorithm applies to Open Multiprocessing (OpenMP) and Message Passing Interface (MPI) applications. OpenMP is used to explicitly direct multi-threaded, shared memory parallelism [4]. MPI standard is commonly used parallel programming paradigm in distributed memory architecture [5].

Our contribution in this paper is as follows:

- Constructed PET matrix for all the possible voltage and frequency levels for different benchmarks to identify the OOP.
- Addressed the concern of reducing energy consumption by proposing energy efficient rescheduling algorithm and demonstrated the energy savings that can be achieved using this algorithm.

The rest of this paper is organized as follows: Section II contains Related Works, Section III describes Optimal Operating Point. Section IV presents the proposed Research Methodology. Section V presents the Experimental Results. Conclusions and remarks on future work are presented in Section VI.

II. RELATED WORKS

There are lots of researches carried out in the field of optimization of power in HPC. In this section, we explored current research works related to energy optimization in HPC clusters. There are various components in HPC system, which participates in power dissipation. CPU is the most power consuming component in a node [2]. The DVFS is used to reduce the clock rate and operating voltage of components which affects the overall energy consumption [6-8].

In order to reduce the power consumption, the frequency of the nodes with less computation has been scaled down by Etinski et al [9] whereas in [10], Dong et al focused on scaling down the CPU frequency during the MPI collective operations. The work in [11] addresses the problem of dynamically optimizing power consumption of a parallel application that executes on a many core CMP (Chip Multiprocessors) under a given performance constraint. In this paper, we have used DVFS technique to build PET matrix in order to compute the OOP for various workloads.

There are some research works which are based on the energy efficient task allocation of MPI jobs. In [1], Y. Ma et al. explain how energy consumption can be reduced by efficient task clustering with task duplication. The paper [12]

discusses how task aggregation can contribute to energy savings. Our technique allocates resources considering OOP as a parameter along with resource requirement to reduce energy consumption.

This paper presents energy efficient algorithm which exploits process check pointing and migration concept. Osman et al. [13] used object migration to achieve load balancing for tightly coupled parallel applications executing in virtualized environments that suffer from interfering jobs, which reduced energy consumption significantly. Process migration technique has been used to provide fault tolerance for supporting continued execution of applications [14]. Our algorithm reschedules the job based on availability of resources and the optimal operating point (OOP) of application to maximize the system utilization.

III. OPTIMAL OPERATING POINT

This section describes the concept of OOP in terms of energy consumption, execution time and performance. OOP for an application is a combination of operating voltage and CPU clock frequency where energy consumption is minimal for a particular execution environment. According to Cameron et al [15], dynamic power consumption is directly proportional to the product of frequency and square of voltage. So any change in voltage or frequency has a large impact on power consumption. This means that running an application at its optimal frequency and voltage level can save power. The motive behind finding the OOP is to minimize the cost of execution in terms of energy. We calculated PET matrix for different benchmarks by repeatedly executing them at all the possible combination of Voltage and Frequency. Based on PET matrix, we can predict beforehand, the required OOP for the application to hit the performance target. We are maintaining a KB for storing OOP of different applications.

A. Experimental Setup

For carrying out experiments, we have used Watts Up? .NET power meter to measure the power consumption of four HP Proliant DL380 G7 (4x8 GB RAM) servers, each having two Intel Xeon E5645 processors with six cores as shown in Fig. 1. Each CPU core has maximum frequency of 2.4GHz and minimum of 1.6 GHz. Each node has RHEL 6.2 operating system and MPI version MPICH2-1.4.1. We have used phc_intel module for scaling the voltage levels. PHC (Processor Hardware Control) is an acpi-cpufreq patch built with the purpose of enabling undervolting on the processor [16]. It works only if the processor's architecture supports undervolting. It provides us with the VID (voltage identification) range of the system. The correct supply voltage is communicated by the microprocessor to the Voltage Regulator Module (VRM) at motherboard via 8 bit number called VID. The VRM is fed by the +12V supply. When the VRM receives the VID, it provides the corresponding constant voltage supply (Vcc) to the processor. For our systems, VIDs lies in the range of 12 to 19. Higher the VID, higher is the voltage provided to the processor. In every system, each frequency level has a default voltage level. For example, in our systems, 1.596 GHz operates on VID=12, 1.729 GHz operates on VID=13 and so on. Using PHC, we can operate a particular frequency on all the VIDs that are lowest than its default VID as well. While conducting the experiments, VIDs were dynamically scaled within the system provided range.

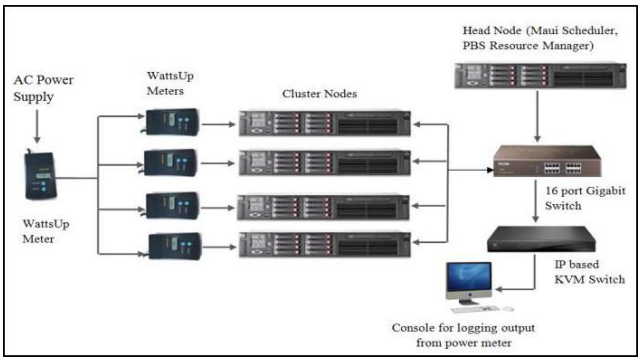

Fig. 1 Experimental Setup consisting of 4 nodes

B. PET Matrix

We constructed the PET matrix for LINPACK [17] and STREAM [18] benchmarks to find out their OOP. LINPACK was executed for 24 number of processes on two nodes (12 processes each) with problem size N=20. Fig. 2-4 shows the energy consumption, execution time and performance results for LINPACK at different voltage and frequency levels.

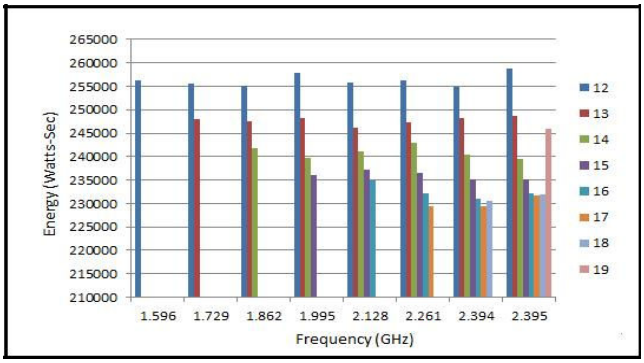

Fig. 2 Energy Consumption (Watts-Sec) of LINPACK

In Fig. 2, X-axis depicts different frequency levels in GHz and Y-axis depicts the energy consumption in Watts-Sec, 12-19 are the different VIDs.

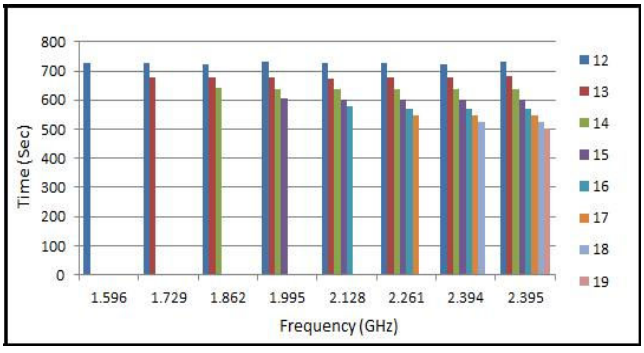

Fig. 3 Execution Time (Sec) of LINPACK

Fig. 3 shows the time taken by LINPACK to complete execution at different frequency and voltage levels. Here, Y-axis depicts the execution time in seconds.

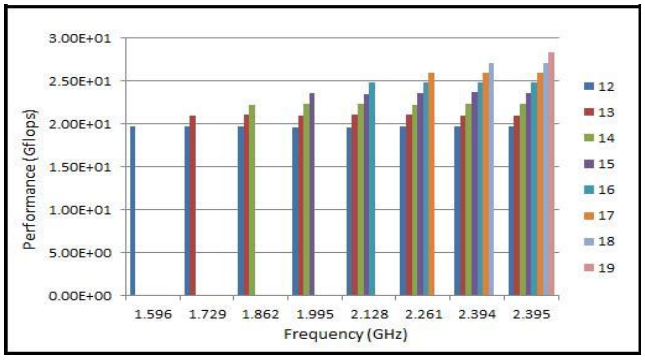

Fig. 4 Performance (GFlops) of LINPACK

Fig. 4 shows the performance of LINPACK at different frequency and voltage levels in GFlops (Y-axis). From the above experimental results, we observed that the energy consumption, execution time and performance rely heavily on voltage. If the voltage level is same, changing the frequency levels doesn't affect application's energy profile considerably. As we increase the voltage, energy consumption and execution time decreases whereas performance improves.

Fig. 5-7 shows the experimental results (energy consumption, execution time and performance) of STREAM benchmark executed for 12 no. of threads. For performing this experiment, we have considered default array size (N = 10000000) as problem size and calculated the computational rate 20000 times (NTimes = 20000).

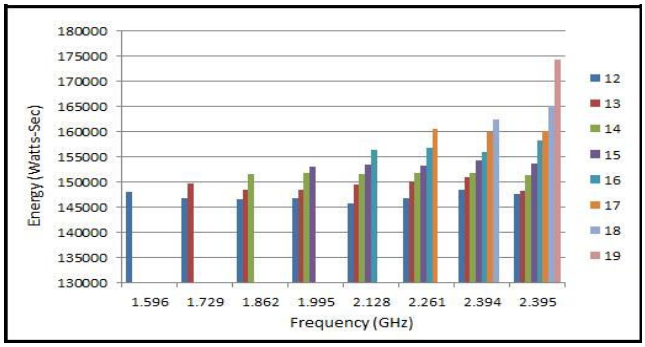

Fig. 5 Energy Consumption (Watts-Sec) of STREAM

In Fig. 5, X-axis depicts different frequency levels in GHz and Y-axis depicts the energy consumption in Watts-Sec, 12-19 are the different VIDs.

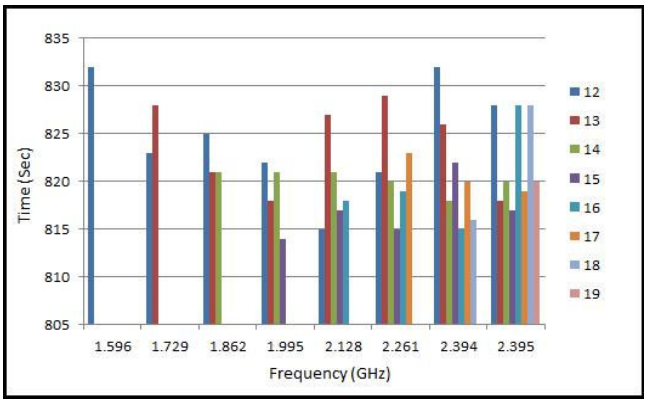

Fig. 6 Execution Time (Sec) of STREAM

Fig. 6 shows the time taken by STREAM to complete execution at different frequency and voltage levels, where Y-axis denotes execution time in seconds. Fig. 7 shows the corresponding bandwidth of STREAM in MB/s (Y-axis). On experimentation, we observed that if voltage is kept constant, changing frequencies have a negligible effect on energy profile. We found that if the voltage increases, energy consumption increases, performance remains almost same and execution time show little bit variation.

Once we found OOP by constructing PET Matrix for an application, example Linpack, we stores that in KB. KB contains details for all the runs of different applications. Prior to scheduling or rescheduling the job, application resource requirement is considered and then the OOP of the application is retrieved from KB. The job is then executed on a node where current frequency and voltage levels meet with its OOP.

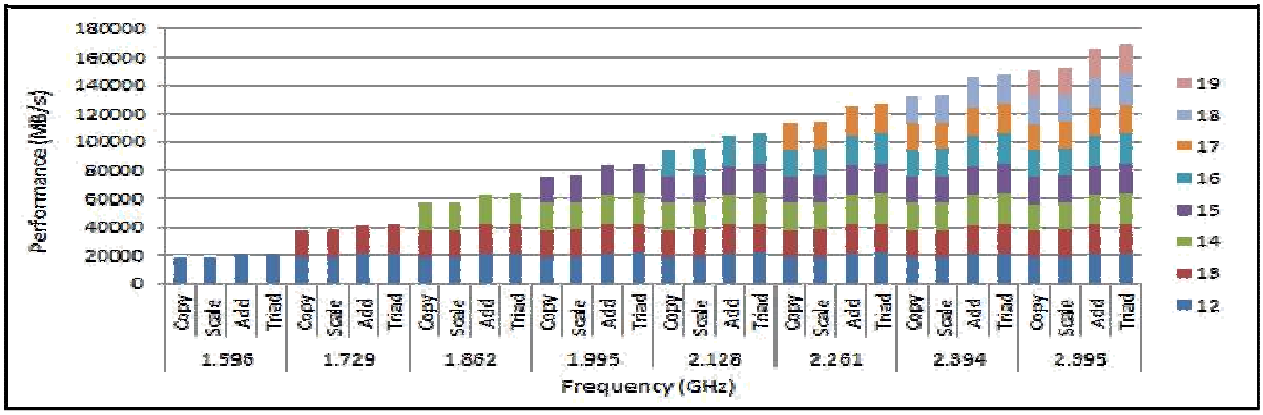

Fig. 7 Performance (MB/s) of STREAM

IV. Research Methodology

This section contains detailed description of proposed algorithm. Here, we explain the working of underlying algorithm with experimental results.

A. Energy Optimization using Energy Efficient Rescheduling Technique

Job scheduling is a critical part of HPC system which provides us an area to reduce the energy consumption. If we dynamically reschedule the jobs within a cluster in such a way that certain nodes can be fully utilized and we can switch off the nodes which are either unutilized or underutilized by migrating its processes to another underutilized node, so that one of the underutilized node can be fully utilized, a significant amount of idle power can be saved. As shown in Fig. 8, if a node becomes idle, we can switch off the node referring the scheduler's job queue.

Our technique dynamically reallocates the processes which are running on a particular node to one of the occupied node whenever it fulfils the resource requirement and satisfy the OOP condition. For example, an MPI job which required four no. of processes (np) is allocated to two nodes with two processes on each (already four cores on Node 1 are occupied as shown in Fig. 8). Our algorithm migrate the processes running on Node 2 to Node 1 whenever cores are free on Node 1 and Node 1 OOP matches Node 2 operating point and switches off Node 2.

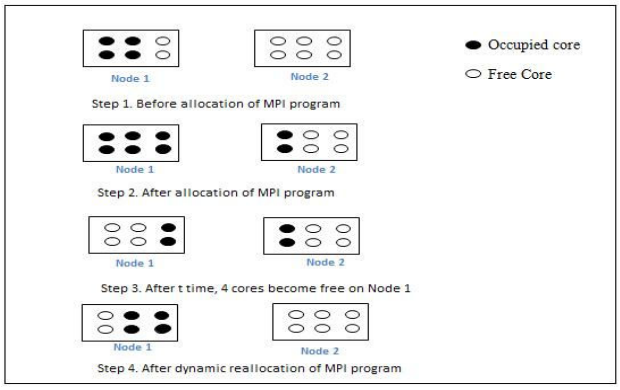

Fig. 8 Dynamic Reallocation of cores

B. Energy Efficient Rescheduling Algorithm

For process check pointing and migration, we have used a transparent user-level check pointing package DMTCP (Distributed Multithreaded Check pointing) [19]. It is a tool to transparently checkpoint the state of multiple simultaneous applications, either multi-threaded or distributed. One of the many applications of DMTCP tool is that it enables a user to run the CPU-intensive portion of the computation on a powerful computer or cluster and then migrate the computation to a single node for later interactive analysis.

In order to migrate a process, firstly we require resource related information to decide whether migration is feasible or not. In that context, we developed a tool which fetches the resource information of the cluster nodes. The tool provides information at real time, displaying the machine name and state, total number of cores available, cores occupied and free. It also displays the jobs currently executing on each node with their Job Id.

After getting the resource related information, next step is to migrate the processes if possible. The energy efficient rescheduling algorithm is shown in Fig. 9. First step of the algorithm is to identify all the systems within the cluster where free cores are available and then store the detailed information (like total no. of cores, no. of free cores and used cores etc.) in a list, sorted based on no. of free cores (in ascending order). The first and last elements of the list are denoted by variables "top" and "bottom" respectively. While the condition (top < bottom) turns out to be true, this algorithm continues to look out for the possibility of migration.

```
Migration_algorithm(node_list)
sort_node_list_based_on_free_core(node_list)
Set top to 0
bottom=length(node_list)-1
WHILE top<bottom:
        node_from, node_to=CALL Migration_feasiblity(node_list[bottom],
node_list[top to bottom-1])
    IF node_to != NONE:
        migrate(node_from,node_to)
        Call Power_off_node(node_from)
        IF index of(node_from)==bottom:
                Decrement bottom by 1
        ELSE:
                Increment top by 1
        Delete (node_from)
        IF node_to->free_core==0:
                IF index of(node_to)== bottom:
                    Decrement bottom by 1
                ELSE:
                    Increment top by 1
                Delete (node_to)
        ELSE:
                Set top to 0
                bottom=length(node_list)-1
                sort_node_list_based_on_free_core(node_list)
                CONTINUE
        END IF
    ELSE:
        Delete(node_list[bottom])
        Decrement bottom by 1
    END IF
END WHILE
```

Fig. 9 Energy Efficient Rescheduling Algorithm

Next step is to check the feasibility of migration as shown in Fig. 10. The node from which the processes have to be migrated is termed as source node (node_from) and the node on which processes are to be migrated is termed as destination node (node_to). The source node is compared with all the other nodes in the sorted list to check whether its processes can be migrated to any other system or vice versa. In both the scenarios, first step is to check the resource availability. All the nodes which fulfil the resource requirement are stored in a list. In case there is no node which fulfils the resource requirement, migration cannot be done.

In the first scenario, where the processes can be migrated from the source node (bottom) to another node, firstly the OOP for all the processes running on the source node is retrieved from KB. Then the OOP for all the processes executing on the possible destination nodes is retrieved and the maximum of optimal operating level of each possible destination node is stored in a list. The maximum OOP of all the processes of source node is compared with the maximum OOP of all the systems present in the list. Whichever node turns out to be the best match, that particular node is selected as the destination node. Before migrating the processes, the maximum OOP of the destination system is compared with the optimal operating requirements of the source node. If the maximum operating point of the destination node is greater than or equal to the required levels of the source node, then there is no change required in operating conditions otherwise the destination node frequency and voltage are scaled to the source node operating level.

```
Migration_feasiblity(node,node_list):
bottom_to_top_list is a List
top_to_bottom_list is a List
ovfs is operating voltage frequency state
ovfs_dest is operating voltage frequency state
node_dest is destination node

FOR item= first to last in node_list:
    IF node's occupied_core  <=  item's free_core AND
    node's jobs_requirement_matches with item:
        Put item in bottom_to_top_list
    ElSEIF item's occupied_core <= node's free_core
    AND item's job_requirement_matches with node:
        Put item in top_to_bottom_list
    END IF
END FOR
IF bottom_to_top_list is empty AND
top_to_bottom_list is empty:
        RETURN (NONE, NONE)
ELSE IF bottom_to_top_list  is empty:
        node_dest=node
        ovfs_dest=max (node_dest's all jobs ovfs)
        node= Select the best match from top_to_bottom_list
        with ovfs_dest
        ovfs= max (node's all job's ovfs)
ELSE:
        ovfs=max (node's all jobs ovfs)
        node_dest= Select the best match from bottom_to_top _list
        based on ovfs
        ovfs_dest=max (node_dest  all job's ovfs)
END IF
IF ovfs ~= ovfs_dest OR  ovfs < ovfs_dest:
        RETURN (node, node_dest)
ELSE:
        increase_vfs of node_dest to ovfs
        RETURN (node, node_dest)
END IF
```

Fig. 10 Migration Feasibility Algorithm

Let us consider second scenario, in which the processes from other systems can be migrated to the bottom node. The first step is to retrieve the OOP for all the processes running on possible source nodes from KB and then storing the maximum OOP of each possible source node in a list. The elements of this list are compared with the maximum OOP of all the processes running on the destination node (bottom). The node which best matches the optimal operating requirement of the destination node is selected as the source node. Prior to the migration of processes, it is verified whether the maximum operating point of the destination node is greater than or equal to the required levels of the source node. If true, then the processes can be migrated without any change in operating conditions otherwise the operating level of the destination node is scaled to the source node operating level.

After migration, the unutilized node can be switched off using the power off algorithm as shown in Fig. 11. But before switching off a node, certain conditions need to be checked. The first condition is to verify whether any job has been scheduled for this particular node. If it comes out to be true, the scheduled time is retrieved from the scheduler. The time for which the system is going to be idle (ΔT), is calculated as a

difference between the scheduled time (T_x) and the current time ($T_{current}$). If the time taken in switching off and restarting the system ($T_{switch\_off} + T_{restart}$) is less than ΔT and the energy consumed in switching off and restarting the system (Energy $_{(Toff + Trestart)}$) is less than the energy consumed while the system is idle (Energy $_{(\Delta T)}$), the node can be switched off. If there is no job scheduled for this particular node, it can be directly switched off.

```
Power_off_node(node):
Tx  = system_needed_for_queued_job(node)
IF Tx  :
        ΔT = Tx     -   Tcurrent
        IF ((ΔT > Tswitch off + Trestart) AND (Energy (ΔT)
        > Energy (Toff + Trestart))):
                power_off node
        ELSE:
                RETURN
        END IF
ELSE:
        power_off node
END IF
```

Fig. 11 Power off Algorithm

In future, if any new job is scheduled then a switched off node can be switched on based on the job's requirement, using power on algorithm as shown in Fig. 12.

```
Power_on_node():
FOR job= first to last in job_queue:
    IF job is waiting for resources:
        system_list= Select system from power_off sys-
        tems where job's requirement == system specification
        Send start notification (system_list)
    ELSE:
        RETURN
    END IF
END FOR
```

Fig. 12 Power on Algorithm

Whenever any new job is scheduled or any running job gets terminated, this algorithm starts again.

V. EXPERIMENTAL RESULTS

To validate the proposed algorithm, we carried out the experiment on a four node cluster. We executed different benchmarks like LINPACK, STREAM and NAS Parallel Benchmarks [20]. As shown in Fig. 13, we conducted the same experiment under two different scenarios i.e. with and without using proposed algorithm. We observed that using energy efficient rescheduling algorithm, a significant amount of energy is saved with some increase in execution time. In our case, we achieved 21.2% energy savings whereas performance (increase in execution time) degraded by 14.4%. The graph shown below depicts the comparison in power consumption pattern of applications executed with or without rescheduling algorithm.

Even though there is an increase in execution time when proposed algorithm is used, a considerable reduction in energy consumption can be observed. Moreover, as the idle systems are switched off, cooling cost also gets reduced. The experimental results may vary depending on the type of application executed, the number of systems that can be switched off and the duration for which the systems are off.

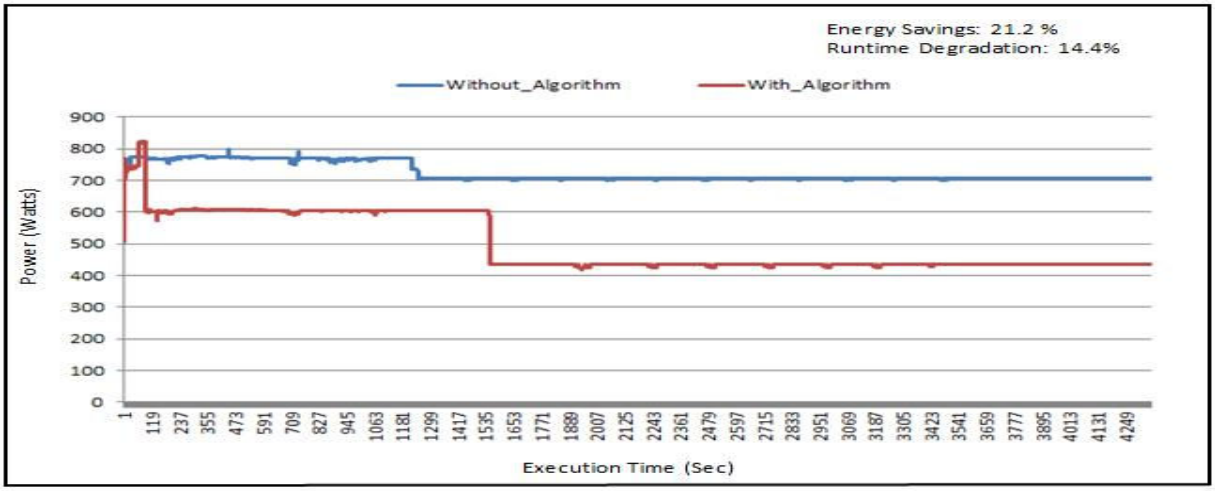

Fig. 13 Experimental results showing overall power consumption of cluster running multiple benchmarks (LINPACK, STREAM and NAS Parallel Benchmarks) with and without using Energy Efficient Rescheduling Algorithm

VI. Concluding Remarks and Future Work

In this paper, we presented an energy efficient rescheduling algorithm in order to reduce the energy consumption in HPC systems based on research, analysis and experimentation. This algorithm relies on efficient resource utilization and Optimal Operating Point. The presented algorithm is illustrated by standard HPC benchmarks like LINPACK, NAS and Stream and respective PET matrix are built and stored in KB. The OOP is devised from PET matrix and used for dynamic rescheduling. DVFS [Dynamic Voltage and Frequency Scaling] and process migration concept are being used in algorithm to reduce energy consumption. The proposed algorithm saves a significant amount of energy. Adapting the algorithm, we successfully carried out the experiments that approve of our approach. We observed 21.2% energy savings and 14.4% degradation in performance at our Test Bed. There is tradeoff between performance and energy consumed which depends upon various factors like type of application, number of processing elements etc.

Our research work leads to various possible directions like power aware rescheduling and dynamic adaptation of optimal operating points (Voltage and Frequency). Further, this work leads to devise an OOP Predictive Model for various target execution environment with varying loads.

References

[1] Ma Y, Gong B and Zou L, "Energy Optimization Scheduling of Task Dependent Graph on DVS-Enabled Cluster System", the fifth Annual ChinaGrid Conference, 2010.

[2] I. Rodero, S. Chandra, M. Parashar, R. Muralidhar, H. Seshadri and S. Poole, "Investigating the Potential of Application-Centric Aggressive Power Management for HPC Workloads," 17th IEEE International Conference on High Performance Computing (HiPC)-2010, Goa, India.

[3] Ryan F, Tyler B, Curt O, Howard JS and Anthony AM, "Analyzing the Trade-offs Between Minimizing Makespan and Minimizing Energy Consumption in a Heterogeneous Resource Allocation Problem," INFOCOMP 2012.

[4] OpenMP Standard http://www.openmp.org [Accessed: May-2014].

[5] The Message Passing Interface (MPI) Standard http://www-unix.mcs.anl.gov/mpi/ [Accessed: May-2014].

[6] Wang L, Laszewski G, Dayal J and Wang F, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS," IEEE symposium on cluster, cloud and grid Computing, CCGrid, 2010.

[7] Li D, Supinski BR, Schulz M, Cameron K and Nikolopoulos DS, "Hybrid MPI/OpenMP power-aware computing," IEEE symposium on parallel and distributed processing, IPDPS 2010.

[8] Cameron KW, Ge R and Feng X, "High-performance, power aware distributed computing for scientific applications," IEEE computer, vol. 38, 2005.

[9] Etinski M, Corbalan J, Labarta J, Valero M and Veidenbaum A, "Power-Aware load balancing of large scale MPI applications," IEEE symposium on parallel and distributed processing, IPDPS 2009.

[10] Dong Y, Chen J, Yang X, Yang CY and Peng L, "Low Power Optimization for MPI Collective Operations," IEEE conference young computer scientists, ICYCS 2008.

[11] J. Li and J. Martinez, "Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors," IEEE High Performance Computer Architecture, 2006.

[12] Li D, Nikolopoulos DS, Cameron K, Supinski BR and Schulz M, "Power-aware MPI Task Aggregation Prediction for High-End Computing Systems," IEEE symposium parallel and distributed processing symposium, IPDPS, 2010.

[13] O. Sarood, A. Gupta and L. Kale, "Cloud Friendly Load Balancing For HPC Applications: Preliminary Work," IEEE International Conference On Parallel Processing Workshops, 2012.

[14] Chao Wang, Frank Mueller, Christian Engelmann and Stephen L. Scott, "Proactive Process-Level Live Migration in HPC Environments," ACM/IEEE conference on Supercomputing, 2008.

[15] Ge R, Feng X and Cameron KW, "Performance-constrained distributed DVS scheduling for scientific applications on power aware clusters," ACM/IEEE conference on supercomputing, 2005.

[16] https://wiki.archlinux.org/index.php/PHC, [Accessed: May-2014].

[17] HPC Challenge Benchmark [Online]. Available: http://icl.cs.utk.edu/hpcc/, [Accessed: May-2014].

[18] STREAM Benchmark [Online]. Available: http://www.cs.virginia.edu/stream/, [Accessed: May-2014].

[19] J. Ansel, K. Arya and G. Cooperman, "DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop," IEEE International Parallel and Distributed Processing Symposium, IPDPS 2009.

[20] NAS Parallel Benchmarks [Online], Available: http://www.nas.nasa.gov/publications/npb.html, [Accessed: May-2014].