2014 Fourth International Conference of Emerging Applications of Information Technology

Learning Based Performance and Power Efficient Cluster Resource Manager for CPU-GPU Cluster

Soumen Kumar Das
Dept. of Space, Govt. of India
Vikram Sarabhai Space Centre, ISRO
Trivandrum, India

Sudhakaran G
Dept. of Space, Govt. of India
Vikram Sarabhai Space Centre, ISRO
Trivandrum, India

V Ashok
Dept. of Space, Govt. of India
Vikram Sarabhai Space Centre, ISRO
Trivandrum, India

***Abstract*— The recent success in building *petascale High Performance Computing (HPC) systems* have produced the demand for efficient and optimized use of resources to increase the performance and reduce the power consumption. Including the above, the heterogeneous architectures of nowadays HPCs comprising a multicore CPU and many-core Accelerator like GPU(s) are facing another concern for using optimum utilization of each of these components. This paper presents the scheduling mechanism of the Cluster Resource Manager (CRM): i. Moldable job Scheduler (MS) which is able to mold the jobs with respect to the number of machines based on an preliminary initialized and auto updated heuristic knowledge-base of problem size, optimum machine count, execution duration to increase the utilization of the full cluster facility. ii) Collocation Aware and Power Efficient Resource Manager (CAPE-RM) manages collocation of CPU only and GPU accelerated jobs by monitoring the CPU load and memory usage. The emerging computation ability is followed by the huge amount of power consumption. Though the use of GPU(s) itself cut down the power to be needed by the only CPU based cluster but to make a *green computing* facility more power efficiency is desired. The CAPE-RM is designed to support the above by powering off the idle nodes by monitoring the total load to the facility and based on a simple statistic of the frequency of job submission.**

Keywords—High performance Cluster; CRM; Moldable Scheduler; Collocation; Resource Manager; petascale; green computing.

I. INTRODUCTION

A. Motivation

The top supercomputers in Top500 list are all capable of petaFLOPS (PF) computation range which points towards the fast growing development of supercomputing technology. The future immense computational power in almost all the facilities indicates the huge number of jobs which can be executed to these modern cluster facilities. There is obviously a demand for an efficient Cluster Resource Manager (CRM) which will be effective in scalable systems and should be capable to handle order of thousands of job in a few seconds. The heterogeneous architectures of computing units multi-core CPUs and many-core GPUs within a single machine, have given rise to a new class of HPCs of more computational power. Achieving the full speedup in these heterogeneous HPCs is difficult as different parallelization architecture is considered. Also from the programmability point of view it's difficult to get the full parallel efficiency. Some popular parallel programming concepts are being developed for utilization of the system e.g. NVIDIA's CUDA [1], an extension to C, has been developed to program General Purpose GPUs (GP-GPUs). Another parallel programming language OpenCL[2] has been introduced with portability feature. High-level interfaces for GPU and/or multi-core programming [3],[4] are being developed. Still the number of efforts in utilizing a combination of CPUs and GPUs to further accelerate application performance is limited [5], [6], [7], [8]. Not only performance but the power efficiency is also nowadays an important concern for the HPC centers. A major fraction of the operational expenditure is spent due to the power consumption and cooling for these HPCs which in turn are harmful to the environment because of the coal based and nuclear power generation.

B. Objective

A Cluster Resource Manager provides services to the incoming processing request, allocate resources and schedules properly. Typically, a CRM comprises a resource manager and a job scheduler. The ultra fast progress of hardware technology is upgrading most of the supercomputers to the petascale range and most of these are following heterogeneous architectures i.e. nodes are made of CPUs, accelerators like GP-GPUs and co-processors. General and typical scheduler cannot achieve the high utilization of resources and overall system throughput of a heterogeneous system as opposed to the need for scaling a single application. The Resource Manager also should be smart enough to reduce power consumption.

Clusters having nodes consists of multi-core CPU and GP-GPUs are the targets for this CRM. The focus is on the scheduling problems related to the delays for scalable systems, load balancing due to splitting into multiple machines,

978-1-4799-4272-5/14 $31.00 © 2014 IEEE
DOI 10.1109/EAIT.2014.58

heterogeneous in nature i.e. CPU only and GPU assisted jobs parallelized by using multi threading, message passing interface (MPI) for multi node jobs and CUDA threading for application of Single Instruction Multiple Data (SIMD) instructions to accelerate a set of applications i.e. mainly Computational Fluid Dynamics (CFD) problems. The primary goal of job scheduling for high performance computing (HPC) is to assign parallel jobs to the corresponding computing nodes, matching the resource requirements of the job. We developed the CRM customized to heterogeneous clusters in such a manner so that it can schedule the heterogeneous jobs i.e. applications meant for CPU only, GPU parallelized application to the suitable node or the set of nodes in case the applications are multiple nodes parallelized. The scheduling problem now involves determining the optimal number of nodes for execution of each application, and whether the nodes can be permitted for collocation or exclusively given to the job. It can also distinguish among the different version of GP-GPUs used in the cluster system and allocate suitable nodes for the jobs based on the size of the problem. Mixing of nodes with different version of GP-GPUs is allowed and the mixing ratio is configurable to optimize. The scheduler of the manager proposed in this work is capable of determining the optimum number of nodes for a job based on its size and duration of execution and past history of node usage provided in a heuristic table. This learning is based on a close loop feedback system which is updating the heuristic knowledge base if any arrived combination is better. The scheduling mechanism depicted above are challenging in nature for a number of reasons. The factors are management of thousands of jobs allowing very less delay, different processing capabilities of CPUs and GPUs, and as a result, each workload performs differently on multi-core CPUs and GPUs [9]. Furthermore, the workload performance can vary w.r.t. the architectural difference of CPU and GPU, the problem size and availability of different version of hardware. Not only most of the CFD applications scalability is getting saturated with the increasing number of nodes but the jobs are getting killed because of the increment of message route and difference between communication bandwidth and computation frequency. So choosing the optimal number of nodes in the moldable scheme is another challenge. Scheduling an application on the node servers where it will perform better, as compared to scheduling it on those that becomes available sooner is a tradeoff.

The Resource Manager of our CRM package is customized collocation aware i.e. capable of allowing dispatch of one or multiple CPU only jobs and one GPU assisted jobs in a single node or set of nodes based on the configuration provided by the administrator by seeing the slowdown ratio. Collocation is possible in between two sets of jobs i.e. CPU only jobs and GPU assisted jobs based on the CPU load mainly and memory. For HPC applications collocation of too many multiple jobs are not a good choice because of the engagement of the processing unit fully. The Resource Manager of the software is designed to be power efficient i.e. the algorithm tends to turn off the idle nodes to save the energy, provided that the system performance is guaranteed by the free left nodes. The system load distribution over a day is the deciding parameter to power off.

The rest of the paper is organized as follows. In Section II, we provide the system Layout, development to the target environment and challenges for the CRM. In Section III, the design and implementation are discussed. In Section IV, the evaluations of the implemented strategies are drawn. In Section V the related works on the scheduler are discussed. In Section VI the conclusions and the future scopes are discussed.

II. System Layout and Challenges

In this section, the *system model*, i.e., the nature of the environment, system architecture and applications being executed are considered. The challenges faced due to the system heterogeneity and application scaling issues are studied and covered.

A. Target Environment

In this paper, we target a cluster of nodes, where some nodes are hybrid i.e. it contains a multi-core CPU and most of them are accelerated with GP-GPUs and some are CPU only nodes. Actually this CRM is designed by taking a reference of the SAGA (Supercomputer for Aerospace with GPU Architecture) [25] of our organization, Vikram Sarabhai Space Centre, India. There could be multiple GPUs on a single node (like the M2090/C2070 GP-GPUs from NVIDIA in the referred cluster). The Hex-core CPU and the GPU are connected through a PCI Express interconnect. Each node communicates with the other nodes in the cluster using QDR (Quad Data Rate) Infiniband interconnection network. Most of the nodes have 24GB RAM and rest consists of 64GB RAM.

The referred cluster and in general most of the cluster are running mainly one instance of identical Operating System. The binaries of the workloads and the corresponding data files need not be communicated across nodes participating because of shared file system for execution of a job. In our execution environment, non pre-emptive execution strategy is followed, and the scheduler can perform on batches of jobs arrive to the system stochastically and are not known a priori. Scheduling decisions are taken based on user inputs regarding execution times and speedups available through analytical models [9] or profiling [13]for existing scheduling techniques [10], [11] and widely used resource managers like SLURM [12].

Applications which need high computational power are the main jobs in these clusters. Some of the applications can either use all cores on the multicore CPU, and some can primarily use the GPUs of a single node or multiple nodes through MPI. In case of GPU intensive applications one CPU thread is still required to issue GPU calls, data copy from GPU device memory to RAM or vice versa and does not consume many computing cycles. It can be assumed that other CPU only jobs can effectively use the rest of the cores scheduled on the same node to use the CPU. The reference cluster is designed to execute the applications which can be categorized as i) GPU assisted job, ii) CPU multithreaded and iii) CPU multithreaded require high memory with having message passing interface to communicate with the other nodes.

B. Strategy Formulation and Challenges

The goal to develop the new CRM on the top of already existing CRM software is to manage the ample number of jobs getting submitted to the petascale system, to utilize the heterogeneous system in both computational performance and power efficient manner. The CRM is running several daemons in a brain server and another daemon is running in all the computing nodes to give node details. The scheduler is efficiently compatible with scalability of the facility with increased requirement of memory linearly and processor demand with the number of job nodes and tasks. The response time increase should be negligible with the system scaling. The *Job Handler* and Queue manager part of the scheduling component should be designed in such a way that they will perform more computation not memory load-store instructions to fetch the correct job for scheduling to the selected nodes which in turn is capable to decrease the response time. Next challenge is to effectively schedule each job on the system based on its required node type. To increase the system throughput molding (choosing the optimum number of machines for a multi-node job) is also implemented to the scheduler. Molding in this CRM can be based on the parameters of each job i.e. problem size, execution duration and number of iterations which will be maintained in a heuristic knowledge base which is being updated in close loop feedback manner. Again maintaining the number of entries based on the range of the problem size and the offset between the entries are another challenge as it is having a direct effect to the implementation efficiency.

Based on the test that exclusive executions of jobs in the nodes are not utilizing the system fully we tried to increase the whole system throughput by increasing the CPU utilization and memory usage with the implementation of collocation of jobs in the nodes. Multiple CPU jobs collocation can be based on the load on CPU of the nodes. The primary challenge is running multiple jobs will slow down the jobs and its turnaround time will be increased. The selections of the jobs which can be collocated with the minimal slowing down of the jobs also need detailed study and observation. Not only the turnaround time but the difference of architecture in CPU, GPU and the communication with other peer nodes are also the parameter of the throughput function. A multi-core CPU has relatively smaller number but quite powerful cores, where Multiple Instruction Multiple Data (MIMD) parallelism can be applied across the cores. On the other hand, a GPU has a large number of simple cores, each of them working in an in-order SIMD fashion. Architectural difference between CPUs and GPUs are their computational capabilities, memory capacities and latencies. GPU programs, hosted by the CPU threads need data transfer between CPU and GPU memory and need to be launched from the CPU. Besides the CPU load the analysis of the computation to memory ratio and of the computation patterns can also help to identify the set of nodes where collocation is profitable or not. The ratio of data copy between CPU main memory and GPU device memory to the computation is also a key factor in deciding whether a CPU workload collocation is suitable beside GPU assisted jobs in the nodes having GPUs. The scheduler may perform scheduling decisions that maximize the aggregate throughput of the workload at the expenses of the execution time taken by each single application job. The scheduler may perform molding, that is, it may assign fewer resources to some jobs than required.

The Resource Manager of the CRM is managing the resources i.e. the nodes status and allocating nodes on exclusive or collocation basis and receiving the nodes back to the nodes pool after completion of job governed by scheduling mechanism. Another challenge in designing Resource Manager is to make it power efficient to reduce the power consumption by the full cluster facility globally.

III. Design and Implementation

The CRM is configurable and the nodes are classified into three groups logically: i) CPU only nodes, ii) GPU accelerated nodes and iii) Nodes with high memory (CPU only) in correspondence with the application nature. The scheduler communicates with the resource manager to obtain information about queues, loads on compute nodes, and resource availability to make scheduling decisions. A detailed description of the working of this package is given below.

A. Job Handler and Moldable job Scheduler (MS):

The Job Handler is designed with *indexed hash table* including *collision-resolution* facility. The design is computation oriented rather than memory consultation thus it is of time constant order. The Job Handler is running in a central brain server dedicated only for management of the cluster. It can handle thousands of jobs without increasing the turnaround time w.r.t. system scaling. The scheduling policies can be set in configuration file based on requirement. Like any standard CRM package multiple scheduling policies can be set over the job queue simultaneously based on the rules set by the administrator. The main constraint is the overhead associated with scheduling should be minimal and within acceptable limits. Advanced scheduling algorithms can take time to run. Improved scheduling performance can be achieved by using the less time complex scheduling algorithm. The implementation of data structure like *look up tables* created is according to the number of scheduling policies running simultaneously. The look up table of the indices of the jobs in job hash table will be sorted parallel based on the criteria of the scheduling policy e.g. priority scheduling, Shortest Job First (SJF) and can be dispatched within a very minimal time. So in this design memory consumption of the central server will be linearly scaled with system scaling which is not a concern but response time will be order of constant. The scheduler will ask the set of nodes based on the requirement of the job i.e. if application is only single node CPU parallel then one node having CPU only, if multimode then set of CPU only nodes. The CAPE-RM based on the availability allocates the single or the set of nodes and the job will be dispatched to the main node (logically assigned) in case of multiple nodes, nodes having GPUs for GPU assisted jobs, CPU only nodes for CPU only job and machines with high memory for the jobs require much main memory on exclusive basis initially.

The advanced feature of the scheduler is molding a job in terms of the number of required machines. Two aspects, the

efficient allocation with a little compromise of turnaround time of job and distribution of resources fairly can be covered in a proper efficient manner with moldable job. The multi-node jobs should be split in such a manner and distributed to each node so that load distribution can be optimal with minimal communication between nodes. The load factor (LF) for each split subtask is deduced from its workload size e.g. for CFD application the load factor depends on the split grid size. Job molding will be based on one heuristic knowledge base $H = <S, T, I, n, \mu>$ containing the previous successful history of jobs. Though initially the knowledge base should be filled by the administrator or trivial case execution will be done but gradually it will be updated based on any better case scenario. The algorithm initially splits the job based on the initial value from H and checks the load factor in each node after submitting.

while ($\Delta LF = LFmax - Lfmin$) ≥ upper threshold {

i. decreasing the no of nodes based considering the max grid size processing capability of each node i.e. f (computing cores, primary memory);

ii. Again re-split the job and resubmit.

}

If the newly assigned node count is getting less then job may be killed due to the high memory usage of one node. Memory usage per node i.e. μ restriction is environment specific if machines are not very much high end. Mainly application parallelized by MPI process has memory cut off for each MPI process. μ of H projects the cut off.

Update the H based on the final node count assigned to the problem, its size and memory usage per node.

The minimal communication traffic can be assured for the job by lowering the number of nodes following the job molding algorithm which in turn provides efficient allocation. Molding also helps to achieve fair share distribution so the number of jobs is getting dispatched will be more and response time for the jobs will be less.

TABLE I. SCHEMA OF THE HEURISTIC KNOWLEDGE BASE

Problem size (S)	***Optimum node count (n)***	***#iterations (I)***	***Execution time (T)***	***Memory usage (μ)***
Grid size/ # computable units cells	No. of machines which is producing optimum load factor	Iteration range	Time taken by the job to be finished with the set of nodes	Problem group wise

Collocation Aware and Power Efficient Resource Manager (CAPE-RM)

It is told that CPU is dispatching the task to the GPUs by invoking the GPU thread task and maintaining the data flow between the GPU device memory and main memory. So any GPU assisted job, running in a set of nodes will not be using the 100% of each CPU core for all the time if the application is not CPU multithreaded properly. That gives the inspiration to collocate a CPU only job to the same set of nodes or its subset. Intuitively, if perfectly coded multiple GPU jobs or too many CPU jobs are collocated together or separately perform worse racing for the resources. The system throughput depends on the tradeoff between the CPU cycles steal and the overhead due to job collocation. CPU is not been used even nearly to (n*100) % having n cores most of the time insists collocation for nearly full utilization. The collocation of job will reduce job execution speed and hence a tradeoff with increment of turnaround time and full system utilization.

Slowdown ratio =

$$\frac{(\textit{Execution time when CPU sharing - Execution time in exclusive mode})}{\textit{Execution time in exclusive mode}}$$

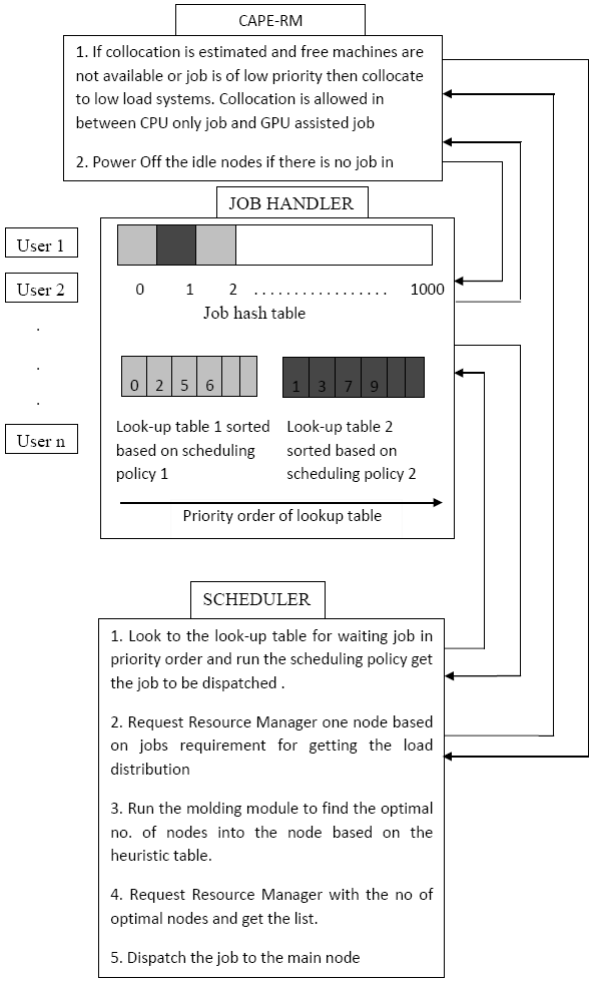

Fig 1. working principle of the CRM

Some scenarios studied in our implementation are covered but thorough testcases are yet to be studied. By default multiple jobs can access any of the cores handled by Linux's default process scheduling algorithm in shared mode. In this design CPU only job can be collocated along with a GPU assisted job in the same set of nodes or subset in an execution timeline of GPU jobs i.e. execution duration of GPU job ≥ of CPU jobs provided by user or taken from the knowledge base can be collocated. The CRM is managing nodes set wise

exclusively for one GPU assisted job and for CPU only jobs (too much CPU consuming jobs are not preferred). The nodes having GPUs will be collocated based on CPU load and memory footprint after allocating the GPU assisted job. The slow down ratio is getting increased if the CPU is getting oversubscribed i.e. each job is using CPU cores more than average (= full core utilization/number of collocated jobs) for a persisting time. Some CPU only applications are very memory intensive so memory footprint is also a criterion for collocation in our CRM. So with a GPU assisted HPC job another CPU only HPC job collocation is not giving the utilization.

Power Efficiency of the Resource Manager is in terms of reduction of power consumption for powering the facility as well as cooling plant. The Resource Manager is monitoring the idle nodes and after consultation with the Job Handler it's powering off the nodes if the system load is manageable by the rest of the nodes without compromising the performance. Thus it's reducing the power consumed for cooling as well as powering the nodes. The energy saving by the strategy

$$E = V*(p-q)*e*t / 100 \quad (1)$$

Where V = total no. of nodes in the cluster, $p\%$ nodes are redundant and $q\%$ of nodes can handle the system load for time t i.e. for t duration $q<p$. e is energy consumption by each node in Watts. The system load can be current load only or statistical average load based on daily time slot. Statistical average load on time slot t will be calculated by an approximation function of the time window configured by administrator.

IV. EVALUATION

The implementation of the collision-resolution based hash table improves the response time for the referenced cluster [25] which is having 0.5 PF peak performances. It can have great effect with the scaling of the system where the retrieval and storing of order of thousands jobs can be done based on hashed function. The heterogeneous scheduling scheme with molding strategy is also achieving more system throughput than the earlier scheme of the centre and it can perform better than other standard typical scheduler. The overall resource sharing is also fair. It is undergoing on detailed tests with training heuristic knowledge base.

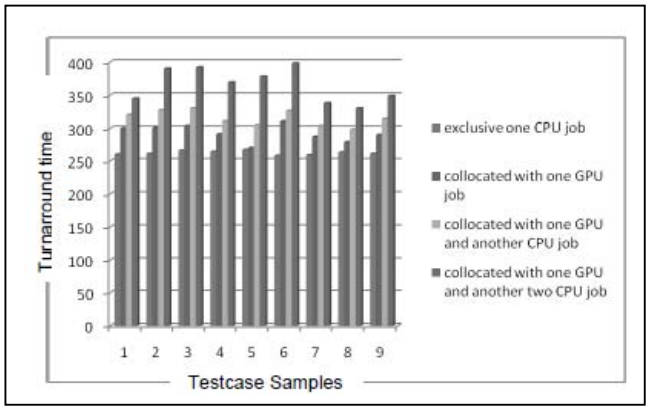

Fig 2. Slow down comparison based on no. of jobs collocated

CFD jobs will be started with the initial values which will be gradually replaced with better parameters running the split algorithm and based on the workload distributed on each node.

The collocation is tested with one GPU assisted job and one, two, and three CPU only jobs respectively. The slowdown ratio is shown in Fig 2 which is affecting more with the more number of collocating jobs. The CPU average load is also going greater than 20 in case of more than four CPU jobs in hex core machine. The GPU job slowdown is nearly 2% w.r.t. the turnaround time.

The power consumption of a NVIDIA M2090 GPU in idle time is 80 watts and 180 watts due to 95% utilization taken at one instance and Intel Xeon E5653 Hexcore processor is 80 watts each which is also will be more than 60% as most of the time it will be executing some system services.

In the reference facility for one node

*Total Power Consumption = (80 * 2) + (80*2) = 320 watts + power required for the other components*

The facility average utilization in terms of node_hour is mostly 95%. About 5% of the nodes are getting idle per day.

Now for any petascale range facility where number of idle nodes on an average can be 50 or more, the energy savings due to powering off the idle nodes and reducing the load on cooling plant can be great aspect. Nodes will be powered on with incoming system load which is continuously being monitored by the Manager. The nodes are getting booted by a light weight operating system where booting time and communication overhead are very less and also power efficient. To save the amount of energy discussed above this overhead is worth doing.

V. RELATED WORKS

In this section, the related work reported in our work of Cluster Resource Manager draw a comparison against relevant work on parallel job scheduling is discussed.

Scheduling of parallel jobs in a homogeneous cluster is a very widely studied problem [14], [15], and [16]. Sabin *et al.* [17] proposed and evaluated job scheduling strategies for multiple sites. Torque [18] is being widely used to manage heterogeneous clusters comprising multi-core CPUs and GPUs. Moldability limitations are there in other related efforts [19], [20] to schedule a job onto a different resource type. Parallel job scheduling with moldability has also been studied [22], [21]. Our work is implementing moldability from an auto updated knowledge base which is still having the inherent issues.

In the context of collocation the workload will be dynamically divided and fed into both CPU and GPU if program dependency is satisfied. Our work is based on overlapping CPU/GPU computation among multiple jobs is implicitly supported by the OS.

Most of the power reduction algorithms consumed focus on the improvement of the single node, by reducing the clock frequency, supplied voltage or by saving interconnect components in computer locally. Some strategies for managing energy in the context of front-end server clusters are proposed [23] [24]. The basic idea is to control the aggregate

system load and then determine the minimal set of servers which could handle the load. Our work is also following the strategy to power on or off on demand but the main computing nodes which are contributing more to the power consumption of the system.

VI. CONCLUSION AND FUTURE SCOPE

In this paper, the increment of delay for scheduling order of thousands of jobs are studied due to the scalable system and the algorithm to provide constant time scheduling is presented. Using a standard data structure (hash table) technique the memory communication is reduced by implanting it to the computation.

We have also considered the problem of optimizing the overall throughput of a set of applications deployed on a cluster of heterogeneous nodes comprising multi-core CPUs as well as many-core GPUs. The proposed scheme is based on different amount of information expected from the user, and focus on different tradeoffs between application wait and completion times with system utilization. Moldable Scheduling scheme (MS) that considers molding of jobs to the number of requested nodes is profitable along two dimensions i.e. the efficient allocation of job with minimal communication and fair share of resources thus better system utilization but with a cost of less greater turnaround time.

The possibility of collocating CPU only job and GPU assisted job on GPU-assisted HPC system to improve the resource utilization is also considered and implemented. Though detailed study is on the progress and a few testcases are been tested but it can be indicated that collocating CPU job and GPU job is a feasible way of improving system utilization. The collocation costs in increment of turnaround time and sometimes failure of jobs also. The future work is to test with various test cases and provide a better solution.

The reduction in power consumption by the CAPE-RM is done by taking the snapshot of the system at the time and powering off the nodes based on the load. Another heuristic parameter can be considered for taking the decision to power off the nodes i.e. the frequency of job submission or the day wise distribution which will give some optimal future picture.

REFERENCES

[1] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *Queue*, vol. 6, pp. 40–53, March 2008

[2] OpenCL, http://www.khronos.org/opencl .

[3] C. Hong*et al,* "MapCG: writing parallel program portable between cpu and gpu," in *PACT '10*, New York, NY, USA, 2010, pp. 217–226.

[4] G. F. Diamos *et al*, "Ocelot: a dynamic optimization framework for bulk-synchronous applications in heterogeneous systems," in *PACT'10*, New York, NY, USA, 2010, pp. 353–364.

[5] G. Teodoro *et al*, "Coordinating the use of GPU and CPU for improving performance of compute intensive applications," in *CLUSTER '09*, 2009, pp. 1–10.

[6] G. F. Diamos and S. Yalamanchili, "Harmony: an execution model and runtime for heterogeneous many core systems," in *HPDC '08*, New York, NY, USA, 2008, pp. 197–200.

[7] C.-K. Luk, S. Hong, and H. Kim, "Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *MICRO '09*, New York, NY, USA, 2009, pp. 45–55.

[8] V. T. Ravi, W. Ma, D. Chiu, and G. Agrawal, "Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations," in *ICS '10*, New York, NY, USA, 2010, pp. 137–146.

[9] A. Kerr *et al*, "Modeling GPU-CPU workloads and systems," in *GPGPU '10*, 2010, pp. 31–42.

[10] G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan, "Scheduling of parallel jobs in a heterogeneous multi-site environment," in *in Proc. Of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing*, 2003, pp. 87–104.

[11] G. Sabin, V. Sahasrabudhe, and P. Sadayappan, "Assessment and enhancement of meta-schedulers for multi-site job sharing," in *HPDC '05*, Washington, DC, USA, 2005, pp. 144–153.

[12] "SLURM: A highly scalable resource manager, https://computing.llnl.gov/linux/slurm/slurm.html.

[13] K. Furlinger, N. J. Wright, and D. Skinner, "Comprehensive performance monitoring for gpu cluster systems," in *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, ser. IPDPSW '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1377–1386. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2011.289.

[14] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel job scheduling - a status report," in *JSSPP*, 2004, pp. 1–16.

[15] A. C. Dusseau, R. H. Arpaci, and D. E. Culler, "Effective distributed scheduling of parallel workloads," *SIGMETRICS Perform. Eval. Rev.*,vol. 24, pp. 25–36, May 1996.

[16] V. K. Naik, M. S. Squillante, and S. K. Setia, "Performance analysis of job scheduling policies in parallel supercomputing environments," in *Supercomputing '93*, New York, NY, USA, 1993, pp. 824–833.

[17] G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan, "Scheduling of parallel jobs in a heterogeneous multi-site environment," in *in Proc. Of* the 9th International Workshop on Job Scheduling Strategies for Parallel *Processing*, 2003, pp. 87–104.

[18] "Torque Resource Manager," products/torque-resource-manager.php.

[19] R. F. t. Freund, "Scheduling resources in multi-user, heterogeneous, computing environments with smartnet," in *Proceedings of the Seventh* Heterogeneous Computing Workshop, ser. HCW '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 3–. [Online]. Available: http://dl.acm.org/citation.cfm?id=795689.797878

[20] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Proceedings of the Eighth Heterogeneous Computing Workshop*, ser. HCW '99. Washington,DC, USA: IEEE Computer Society, 1999, pp. 30–. [Online]. Availablehttp://dl.acm.org/citation.cfm?id=795690.797893.

[21] W. Cirne and F. Berman, "Using moldability to improve the performance of supercomputer jobs," *JPDC*, vol. 62, no. 10, pp. 1571–1601, 2002.S.

[22] Srinivasan, V. Subramani, R. Kettimuthu, P. Holenarsipur, and P. Sadayappan, "Effective selection of partition sizes for moldable scheduling of parallel jobs," in *HiPC*, 2002, pp. 174–183.

[23] J. Chase, D. Anderson, P. Thackar, A. Vahdat, and R. Boyle, "Managing Energy and Server Resources in Hosting Centers", *Proceedings of the 18th Symposium on Operating Systems Principles*, October 2001.

[24] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath, "Dynamic Cluster Reconfiguration for Power and Performance", *Compilers and Operating Systems for Low Power*, Kluwer Academic Publishers, August 2003.

[25] Sudhakaran.G, Thomas.C.Babu and Ashok.V, "A GPU Computing Platform (SAGA) And A CFD Code On GPU For Aerospace Applications" in *Proceedings of the ATIP/A*CRC Workshop on Accelerator Technologies for High-Performance Computing: Does Asia Lead the Way?* Biopolis, Singapore, ACM 2012, pp-117-121.