Power-Efficient HPC Clusters

Ramya Rao
Indiana University
Bloomington, Indiana
ramrao@umail.iu.edu

Vaishnavi Mukundhan
Indiana University
Bloomington, Indiana
vaismuku@indiana.edu

ABSTRACT

Maintaining lower power consumption rates in HPC (High Performance Computing) clusters is attracting a lot of attention nowadays. Power management is crucial and essential in HPC clusters, which otherwise can severely affect the performance of the clusters. In this report we explore three approaches taken to judiciously use power in HPC clusters. We analyze each method and present their advantages and disadvantages.

KEYWORDS

HPC, clusters, Virtualization, Dynamic Voltage Scalling(DVS)

1 INTRODUCTION

Clusters usually consist of many MPP(Massively Parallel Processors) which are also referred to as nodes and is self-sufficient. A node has a CPU(Central Processing Unit), I/O subsystem, memory, operating system etc., which is essential for a super computer. All the nodes in a cluster, are capable of communicating with each other through the I/O subsystems present in them. A HPC cluster, contains many nodes with super computing abilities as described above. But, HPC clusters are tailor made to run application which require massive computations and are time bound. Advances in hardware technology as given rise to cost-effective, parallel processors to be incorporated into this HPC cluster.

Figure 1 borrowed from [6] shows the architecture of a fundamental HPC cluster. There is a head node or the manager node, which is connected to the compute nodes or worker nodes. Typically, jobs or applications which have to be run in the HPC environment are queued in the head node, which assigns the compute nodes with tasks of running these computations. HPC clusters are used in a wide range of applications such as life science researches, computer vision research, oil and gas exploration to name a few.

One of life science researches is to reconstruct protein sequences using human gene, which have to be rendered in 3-dimensions to view the folds in the shape. The fold formation analysis helps in detecting diseases such as Alzheimer's etc,. Since the number of shapes, in which it has to be rendered can be potentially infinite and also the protein sequence can be as long as a couple of hundreds of data points. Hence, this analysis can plausibly take a long time to finish the computations and might require parallel computing to achieve this in a reasonable amount of time. Another application to showcase the usability of HPC clusters is analyzing terabytes of information obtained by seismograms. Seismograms collect data about the interior temperature of the earth, formations etc., which can support oil and gas explorations and other forms of explorations. From these examples, we can deduce that, the role of HPC clusters can prove to be imperative for high performance computing environments.

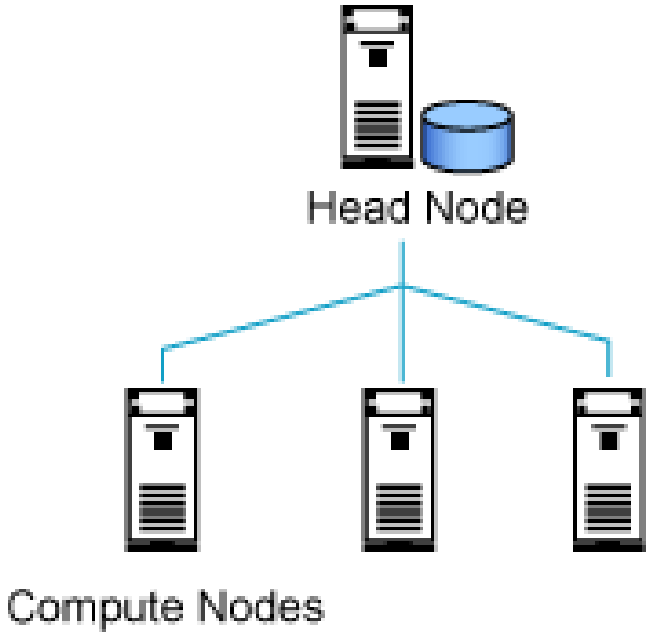

Figure 1: Architecture of a basic cluster

HPC clusters as discussed, use parallel computations to run advanced applications reliably and quickly. Due to the advances in hardware technologies, GPU's can be used to perform general processing tasks. With the incorporation of GPU's in HPC clusters, the performance gain is immense. However, there is also a caveat associated with this improvement, which is the increased power consumption rate in HPC clusters.For instance, NVIDIA Tesla C2050 compute-optimized GPU consumes up to 225 watts[5]. It is a matter of concern to note that, in HPC clusters and data centers, there is 50% more energy consumption. Since chip power efficiency is not improving at historic rates as before. For example, Google uses about 2000 nodes in a cluster to perform Map Reduce operation[1]. This suggest that, there is an increase in the number and size of HPC clusters used today. Hence, it seems appropriate to dedicate research efforts to understand the implications and explore ways to alleviate the issue of increased power consumption in HPC clusters. Reducing the power consumption is important for two main reasons, systems reliability and cost of operation. In this paper, we comprehend, three different ways to address the afore mentioned problem.

Firstly, with the aid of virtualization, [2] show that, we can multiplex a CPU cluster to make use of GPU nodes through a server. This reduces the power impact on the CPU cluster. Since virtualization, paves way for the CPU clusters, to make use of the

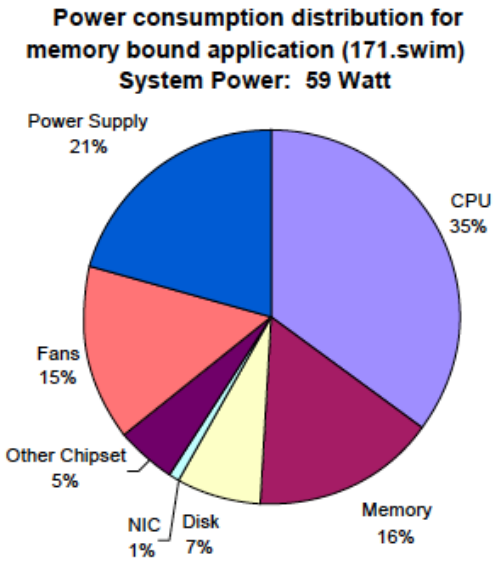

Figure 2: Power consumption distribution for model based on Pentium III

large pool of GPU nodes, without having to deal with the increased power consumption rates. [2] describes and uses rCUDA, which is a framework to efficiently virtualize GPU access to CPU clusters.

Secondly, [8] introduce numerous power models to dynamically distribute the load on all the CPU on a HPC cluster. There is a large variance associated with the, application input sizes, memory footprints and also the execution rates of different applications, etc. Assess the idle times of different nodes in the cluster and effectively use that information. These facts can be inspected in great detail, to learn optimal placements of applications in the available pool of nodes in a cluster to devise a power aware system.

Finally, [3] takes a hardware approach to solve the problem of increased power consumption. As shown in 2, CPU consumes the most amount of energy as compared to the other modules [3]. Thus the hardware solution provided in this paper focuses on reducing the energy used by the CPU there by making the system power efficient.

2 BACKGROUND

Although there are several research papers which address power consumption aspect of HPC clusters, we discuss, few very close ideas with respect to the papers that we analyzed.

Work by [4] was most related to the paper that we researched [2], which virtualized a single GPU across to multiple CPU clusters using a management mechanism called Blink. This paper hugely influenced [2] in designing a system by scaling the number of GPU's available to CPU's from one to many.

Chromium introduced by [5] is another framework which multiplexed multiple GPU's to accomplish rendering task very efficiently. This paper also concentrated on the communications protocols and systems to establish reliable communication between the head node and worker nodes, which consists of GPU's. This asynchronous

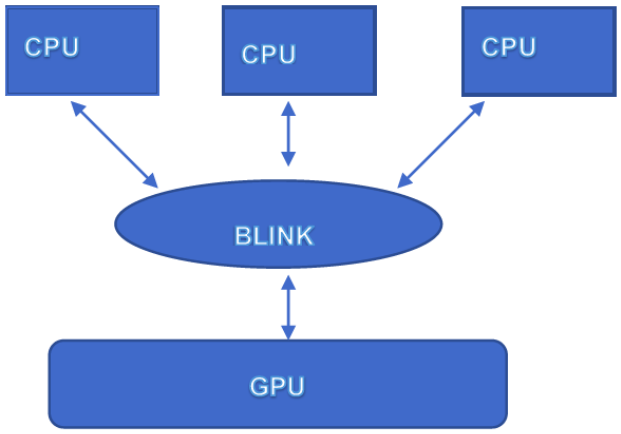

Figure 3: Blink

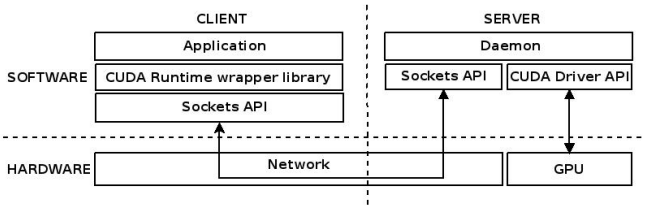

Figure 4: rCUDA framework

memory sharing and also I/O protocols used influenced [2] in devising the rCUDA framework.

pMapper by [7] introduced an architecture which built a power-aware management system to optimize the power requirements in HPC clusters. Work by [8], which we analyze, extends the work by [7] by additionally including performance/workload manager and other enhancements to improve the power gains obtained in HPC clusters.

As shown in Figure 3, Blink[4] is a system which safely multiplexes multiple untrusted CPU data processing requests to a single virtual GPU. The system intervenes between the CPU and virtual GPU and streamlines the requests to effectively utilize the full power of the GPU.

3 APPROACHES

3.1 Virtualization Approach

Figure 4 is the pictorial representation of the rCUDA framework developed by [2]. CPU's which are also called clients, are systems, which accesses GPU's to run heavy compute-intensive work units. All the clients can see, any of the CUDA supported GPU's present in the framework. When a client requires, GPU access, user from client, has to write the unit which has to be run in a GPU in CUDA C language also known as kernel. This code is intercepted by the CUDA Runtime wrapper library in the client side. The linker loader in the client system, automatically loads the wrapper library whenever a kernel is sent to it. The loaded library establishes a reliable connection with the server using the API socket.

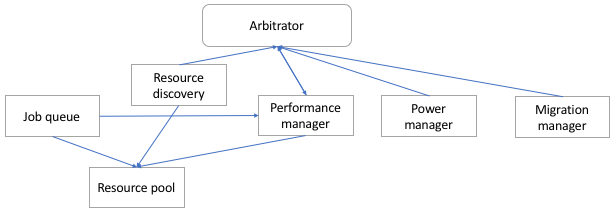

Figure 5: Dynamic placement architecture

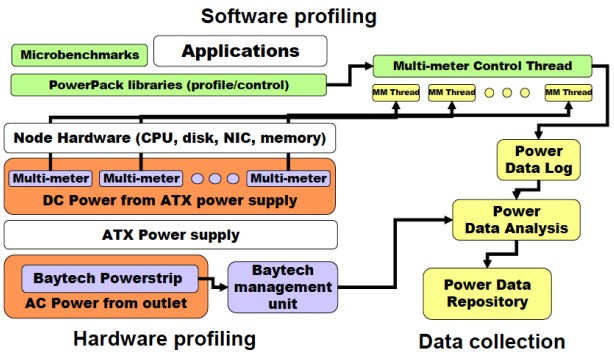

Figure 6: Power-Aware Cluster

Server side has a daemon process continually scanning for requests from clients. Once, a request is received, the server redirects, the piece of code which has to be run on the GPU by running it through a CUDA Driver API. As soon as the execution is complete in the GPU, the results are communicated back to the client from where the request was originated.

Multiple GPU's are provided for the clients by spawning different server processes for each remote execution over a new GPU context. This offers insulations, in the event of crash of a server, so that only the affected server has to perform recovery process. Managing memory amongst the working GPU co-processors can help run all the processes, concurrently.

The operating system scheme for executing client threads are FIFO (first-in-first-out order). As the server is not programmed in software or provided with hardware abilities to determine the level of importance of a request from the client or, distinguish between clients for GPU access. The FIFO ordering makes things simple and easy for the framework to handle.

3.2 Statistical Approach

Figure 5 is the architecture published by [8] to dynamically place jobs in the resource pool. The resource pool consists of CPU's as cluster which are the the worker machines for computations. Arbitrator is the main intelligence in the system which consolidates the inputs received by power manager, performance manager, Migration manager, Resource discovery and makes informed decisions about running applications in the resource pool.

The Migrations manager, is responsible for calculating the time required to move, applications from one CPU/GPU to another and also moving applications from one CPU/GPU to another. Power manager, estimates the idle times of the available resource pool to infer which CPU/GPU is under utilized. Performance manager records the performance of the resource pool on the jobs that are presented to them based on the time to completion etc. The arbitrator compiles all the information influx and instructs the resource discovery for effective utilization of the worker nodes in the resource pool.

The paper also introduces sorting applications submitted to the job queue intelligently, by segregating them based on the working set size of the application. By conducting a large number of experiments to analyze the power consumption, memory utilization and other aspects, by running, different applications with various working set sizes, on the resource pool they conclude that placing similar working set sized application together, would provide huge performance gains. Hence, they divide applications which fit within the cache size of the server as category 1, applications with working set size greater than that of the cache server size are categorized as category 2.

Category 1 applications are tuned to provide no performance degradation and category 2 applications are designed to provide power savings. They also have category 3, to add flexibility to place applications either in category 1 or 2 based on the status of execution.

3.3 Hardware Approach

This paper focuses on building power efficient systems for highly parallel scientific applications that run on distributed clusters of GPU. it takes advantage of distributed clusters that contains multiple power modes. The main idea behind using Dynamic Voltage Scaling (DVS) is DVS would reduce the power consumption of the CPU by fluctuating the voltage supply. We use DVS because it reduces the energy consumption without affecting the execution time. DVS makes use of the delays or shot duration of time when the CPU is idle and modifies the power consumption.

In distributed systems, CPU is underutilized due to various reasons. Delays or CPU idle time occurs due to memory coherence problem or if a process is waiting on an input. At such times the CPU's power mode is changed and it runs on lesser voltage mode, thus reducing the power consumption.

We can schedule DVS in either External or Internal mode. External scheduling of the DVS can be done from the command line or by using Daemon processes. External scheduling using command line is easier to implement. Internal scheduling, uses source-code as an instrumentation to modify the performance or the power modes. It is the most flexible and adaptable method to implement DVS strategy.

Figure 6 shows a framework for the PowerPack module. Using this we can measure and control the power supply in the Power aware clusters. Here the hardware and the software applications are connected to the PowerPack, which does power data analysis to make the distributed clusters of nodes Power efficient.

GPUs	Consumption	Savings	Rate
100	80622.0 W	0.0 W	0.0%
50	70943.0 W	9679.0 W	12.0%
25	66103.5 W	14518.5 W	18.0%
10	63199.8 W	17422.2 W	21.6%

Figure 7: Power usage results from [2]

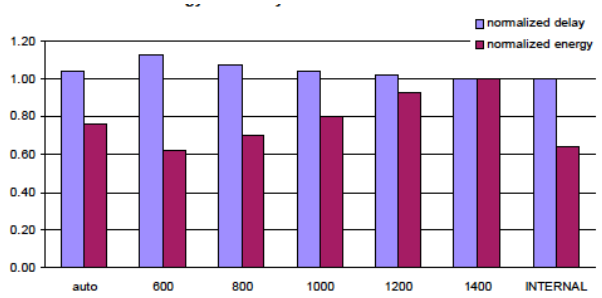

Figure 8: Power usage of External vs Internal scheduling results from [3]

4 ANALYSIS

4.1 APPROACH 1

Paper by [2] presented a very elegant way of multiplexing CPU clusters with high performing GPU access using virtualization. The main goal of reducing power overhead on the CPU nodes due to the use of GPU is reduced to a significant amount by this method. The results shown in Figure 7 shows the power savings by multiplexing different number of GPU's on a 100 node CPU cluster. All the experiments were performed in Quad-Core Intel Xeon E5520 processors running at 2.27 GHz and 24 GB of main memory. Nodes were running Linux OS and the GPU processors were NVIDIA Tesla C1060 (driver 190.18) attached to a PCIe 2.0 x16 port.

The table 7 shows that, the amount of power savings from rCUDA framework starts to decrease to 0% when the number of CPU's in the framework is equal to the number of GPU's. They get a peak 12% reduction in power while using 10 GPU's in the framework. The trait of power savings reducing as the number of GPU's increase in the framework is attributed to CUDA library not being open source.

The paper also changes few of the library functions from CUDA source files in order to then the experiments to show the desired results. This requires modification in the standard CUDA library, which seems tedious. Crash recovery is not addressed in great detail in this paper, which is an important topic for obvious reasons. Since hardware is prone to wear and tear, rCUDA framework would benefit from incorporating logging or such mechanisms to ensure reliable computations in the framework.

4.2 APPROACH 2

It is interesting to see a paper which uses strategic techniques, to obtain power savings which is the reason behind choosing this paper. [8] study the HPC environment using IBM HS-21 Bladecenter with X-series blades. The blades use an Intel Xeon 5148 quad-core processor, with 2.33 GHz core frequency 4MB L2 cache. They use fedora 6 as their guest operating system. They use a variety of workloads for they experiments.

Based on experimentations, they show that there is a dynamic range in the variability of number of jobs submitted to the system with respect to time by recording the idle time of worker nodes over a period of 30 days. This leads to utilizing worker nodes effectively by placing jobs to maximize idle time among all the worker nodes and minimize heavy loads targeted to a single worker node. This result, paves way for a lot more research in the area of dynamic placement of application, by fine tuning the arbitrator to suit the requirements of the system and obtain optimal savings in terms of power.

However, all the gains seem to vanish with increase in the number of applications than the number of nodes available in the resource pool. Also, the experimentation is done with very few compute nodes, that is they use around 11 nodes in the resource pool. This might not be sufficient to prove the mettle of the results implicated by this paper.

4.3 APPROACH 3

This paper presents a novel technique to reduce power using hardware resources for highly parallel scientific applications. Even though there are a few state-of-art hardware systems to save power on distributed clusters, the approach used in this paper is unique. This is because using DVS reduces power without affecting the efficiency of the system.

I found the experiments performed in this paper particularly interesting because they tested the system on different parameters. The power saving across these parameters like workload, application, DVS scheduling strategy and system remained consistent. Total energy saved by this method is about 36%. Another, interesting experiment from this paper showed us that the most flexible methods might not always lead to better solution in comparison to simpler methods. This process needs to be automated to be more adaptable.

Figure 8, shows the experimental results between energy savings and delay in time. We can see that the internal scheduling scheme can save 36% of energy without much of delay. Where as, External saves 38% with a time delay of 13%.

In my opinion, one disadvantage of this implementation is that it is a very manual process. User does not have much control over how the power modes are changed. If user is given more control over the power modes, better savings in energy can be made as many a time, there need not be time delays to save energy.

5 CONCLUSIONS AND FUTURE WORK

Power efficient HPC clusters are very important to improve the system reliability and cost of operation [3]. We have explored three

different approaches (Visualization, Statistical and Hardware) to build a power efficient HPC clusters. These three methods exploit the weakness of the existing system to build a better and a more efficient system. All the three methods have their own share of pros and cons.

The experimental results for the three methods shows a promising gain in power savings while maintaining the efficiency of the system. We have also discussed the few minor changes to the three approaches that would lead to improved performance in saving power.

It would be interesting to see an implementation that uses a combination of two or all of the three methods together. This would exploit the weakness of the three strategies and give us a better solution for power efficient HPC clusters.

REFERENCES

[1] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[2] José Duato, Antonio J Pena, Federico Silla, Rafael Mayo, and Enrique S Quintana-Ortí. 2010. rCUDA: Reducing the number of GPU-based accelerators in high performance clusters. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on*. IEEE, 224–231.

[3] Rong Ge, Xizhou Feng, and Kirk W Cameron. 2005. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*. IEEE, 34–34.

[4] Jacob Gorm Hansen. 2007. Blink: Advanced display multiplexing for virtualized applications. In *Proceedings of NOSSDAV*.

[5] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D Kirchner, and James T Klosowski. 2002. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM transactions on graphics (TOG)* 21, 3 (2002), 693–702.

[6] Aditya Narayan. 1999. Clustering fundamentals. (1999). https://www.ibm.com/developerworks/library/l-cluster1/index.html

[7] Akshat Verma, Puneet Ahuja, and Anindya Neogi. 2008. pMapper: power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*. Springer-Verlag New York, Inc., 243–264.

[8] Akshat Verma, Puneet Ahuja, and Anindya Neogi. 2008. Power-aware dynamic placement of hpc applications. In *Proceedings of the 22nd annual international conference on Supercomputing*. ACM, 175–184.