ISSN 2278-3091
Volume 9, No.3, May - June 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse159932020.pdf
https://doi.org/10.30534/ijatcse/2020/159932020

Dynamic Allocation of High Performance Computing Resources

Manish Kumar Abhishek[1], D. Rajeswara Rao[2]
[1]Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, India, manish.abhishek@gov.in
[2]Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, India, rajeshpitam@gmail.com

ABSTRACT

With the increasing demand of fast computation, High performance computing is getting evolved now a days rapidly. In Today's data era where problems are going to be huge and need high computational resources, High performance computing (HPC) is emerged based on multiple system parallelism equivalent to facilitate supercomputing functionalities. It is going to have a cluster of parallel interconnected computational systems to reduce time of the execution. Several tools are available for cluster management, but resource usage and consumption are always a concern. This paper presents a model which will help in dynamically allocation of the computational resources in terms of CPU, memory to overcome the existing challenges and reduce performance overhead. The results focus on the different configuration for computing resources and comparison of time entities for execution with different number of interconnected processes.

Key words: Computing Resources, High Performance Computing, Resource Manager, Parallel Computing, Resource allocation, Torque, Virtualization.

1. INTRODUCTION

In today's data-centric world, High Performance Computing is always in demand to solve large complex datasets problems. These are going to be multiple parallel running jobs running with static computing resources allocations using well known tools like Message Passing Interface (MPI). The underlying infrastructure should be enough capable in terms of memory, CPU, GPU, network bandwidth and much more to facilitate supercomputer functionalities. Available HPC resource managers like Torque, resources will be assigned to running job based on the specified configurations. If there is any change required in allocated computing resources, the whole job needs to be cancelled and need to submit it again in queue due to its static behavior in nature. Therefore, we need dynamic allocation of computing resources that will not only increase the performance of running application but also enable the Data Intensive application to run on HPC cluster.
HPC applications usage are on boom in research area, IT industry as well as in educational institutes to solve the real time issues. HPC cluster formation is higher in cost but now a days it can be reduced easily via using commodity-off-shell hardware components or leveraging the virtualization techniques [1]. Linux Operating System is most preferable for building the interconnected systems in form of clusters. Parallelism is an essential key performance factor to run HPC applications.
In parallel computing, the issue is split in multiple parts as a series of instructions which can be solved parallel on multiple running processes within a cluster and results it in better performance and execution time. Generally, within an HPC cluster, running jobs are not going to consume full resources but still we need to take care of resource allocation wisely. These free resources can be allocated to other applications like we mentioned above that is Data intensive applications. A little relocation of any computing resource in HPC cluster can highly impact the performance of whole running application, we need to take care of allocation of free resources for data intensive applications very carefully via maintain a buffer. Typically, in HPC cluster same pattern is getting used for running the jobs multiple times with a different set of Input data. Using monitoring and profiling technique, computing resource can be defined within a specified range. In this paper, we represent a model for the dynamic reallocation of computing resources based on running HPC application profile and execution time patterns.

2. RESOURCE ALLOCATION APPROACHES

2.1 Resource Allocation Managers

HPC resources can be managed by using any well-known resource manager where users can describe the jobs by defining their deadline and configurations like CPU, number of threads based on core, memory etc. Running job is not generally going to consume all resources. The allocation of resources is typically based on the granularity of computing node. The computing nodes can be shared across jobs. Overall, Job can be categorized mainly in five types i.e. adaptive, malleable, rigid, moldable and evolving. Adaptive jobs are dynamic in nature which are responsible for large datasets problems and can be adaptable on resource allocation change requirements. Rigid jobs require the detailed format of

resource constraint before job submission and used for long running applications.

A. Torque

It is a well-known distributed resource manager which is used to control over the jobs and distributed computing nodes [2]. It handles the fault tolerance, scheduling interface with high logging, collection of data once job get completed and significantly handle larger clusters to achieve scalability. It is getting used in research supercomputers. Commands that are generally getting used are:

- Qsub – It is used to submit a job
- Qdel – It is used to stop the job before its completion.
- Qstat – It is used to check the current status of scheduled job.

B. Slurm

It is one of the most used open source resource managers for clusters having Linux Operating system. It is almost self-contained and do not require any kernel modification to be operational [3]. It starts with the allocation of both types of exclusive and/or non-exclusive access to compute nodes resources. After allocating the resources, it starts, execute and monitor the work via managing a queue for the remaining tasks.

C. Mesos

It is a cluster resource manager which provides isolation and resource sharing among distributed jobs or applications. Using it, multiple applications can run via sharing the computing resources of nodes within cluster at runtime. It is basically using the cgroups feature of Linux OS for providing the isolation [4]. It offers multiple APIs for executors and schedulers like resourceOffer based on offered and offers. It also reduces the overhead of performing the manual steps for application deployment and automated workload shifting to achieve fault tolerance.

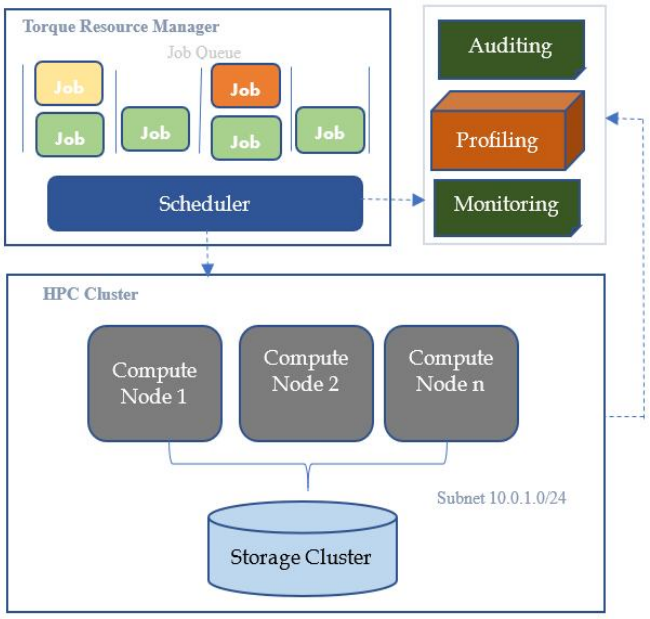

Figure 1: General Resource Management with Torque

Figure 1 shows the general mechanism of managing the computing resources in HPC cluster where jobs are typically queued, and resources will get allocated once get freed. There is no priority assigned to running jobs based on which resource will get allocated. In HPC cluster, resource allocation is always a challenge where distributed jobs are running with different configuration of resources on computing nodes. For this kind of scenario, dynamic resource managers are used to have the fault tolerance. In general, HPC clusters, allocation of resource is done via static configuration of CPU cores, memory, I/O threads etc. Co-scheduling is one of the most used technique to make a change in resource utilization which can raise SLA violations. For performance, Paragon classify each job based on the weightage of resource impact to select the appropriate candidate to collocate the job.

2.2 Challenges in Resource Allocation

In large HPC clusters, due to lower computing resource fragmentation, fine granularity in task allocation can have negative impact. Therefore, we should have a method for resource allocation across jobs. Using the available resource managers, it's hard to integrate the complete analyzed data for scheduling of jobs and control over computing resources where it should be operated with isolation [6]. Deployment infrastructure and its efficient use is also one of the challenges in front of HPC. The usage of free computing resource from the hosted environment is one of the major concerns. HPC application requirements are getting complex, changing day by day and to handle all, we are proposing a model via predicting the running application profile and execution time patterns. It is scalable, flexible in nature and results are also promising for dynamic allocation of resources at runtime.

3. PROPOSED MODEL FOR DYNAMIC RESOURCE ALLOCATION

Here, we are defining the design of out model using which computing resources within a HPC cluster can be allocated at run time to achieve the fault tolerance, isolation as well as meantime available resources can be used for other type of applications. Figure 2 shows system component diagram of our model having main components and complete execution from queue to completion of a job that is managed by Mesos which is non-intrusive and uses cgroups. In Unix, cgroups is one of the easily available and user-friendly operating system controls which ensures that running process consume less resource capacity in terms of CPU, I/O, memory, disk space) from the allocated ones. We have used virtualization to setup the HPC cluster using interconnected virtual machines (VMs). KVM hypervisor has been used to provision the Virtual computing nodes. As we have chosen Linux Operating System (CentOS 7.0) for the deployment of HPC applications, hypervisors also support Linux as a host Operating system as well as guest operating system. For HPC cluster, it is very important to choose virtualization

technology very carefully as it has its own critical aspects. Hypervisors ease the provisioning of virtual machines. Core Service is its one of the core components that we have implemented via following the microservice architecture instead of monolith to achieve scalability and fault tolerance.

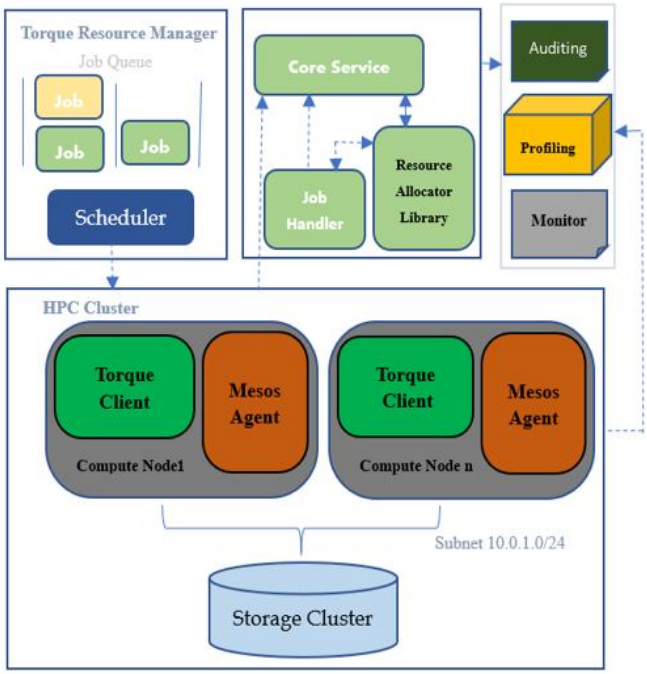

Figure 2: Proposed Model Architecture in HPC Cluster

A. Core Service

Application will be deployed in and initialized by the core service. It calculates, analyses the free computing resources and persisting the same in database. It will gradually learn the pattern of the resource consumption and build a model for same. Mesos is going to use the same. The same list of resources can be shared with the master node to start non-HPC application execution. The profiling will get start once job will get submitted. The profiler is going to hold the performance counters which includes, Core usage, memory, CPU, cycles per instruction. We are going to show all to user to make aware about the resource consumption and can identify the hotspots during a duration of time like past 2 hours. The threshold can be set for the allocated resources so that events will get generated in case of threshold breach. Internally, we have used the microservices architecture for this model. All the node data is going to be consumed by this service to estimate the standard deviation, average for co-scheduling and outliers as well. This service is very light weight in nature having no memory footprints.

We have designed the Algorithm 1 which is based on to choose the appropriate node for container allocation. For each computing resource factor, we have computed its affect on the type of service and rank it. We have the CA_s as the selected allocated container and Service type as the weight vector which gives more wightage to each impacting factor. Computing resource factor includes cpu, memory, affinity, network and GPU. Here overall ranking will be calculated six times of m that means for the computation of allocation of n containers, it will be computed based on 6 multiply by m and n times with a complexity O (n x m).

Algorithm 1. pseudocode for the container placement

Input: C_n, W_n, S_t
Output: C_p
1: $CA_s = \varphi$
2: **for each** container $\in C_n$ do
3: initialize CA_s with N_s
4: **for each** m $\in N_s$ **do** W_n
5: $Rank_m = \varphi$
6: calculate Rankcpu, Nc
7: calculate Rankmem, N_m
8: calculate Ranknet, N_s
9: calculate isGPU, t, f
10: calculate Affinity, t, f
11: Ranktotal, $N_* = Rank_N * S_t$
12: **set** CA_s with $Rank_N$
13: **end for**
14: **sort** CAs with node_Total_score by ASC
15: **map** (container, CA_s [0]) **into** C_p
16: **end for**
17: **return** C_p**;**

- **C_n**: Required containers need to be created
- **N_s**: Available Working Nodes
- **S_t**: Service Type
- **CA_s**: Available node for container deployment
- **$Rank_N$**: The Rank of Container for deployment

4. EVALUATION

Here, we are evaluating our proposed model for dynamic allocation resources, usage of free computing resources for non-HPC application, performance, profiling, queue throughput. We have executed two HPC application and one non-HPC application with static as well as dynamic use cases. The three cluster sizes have been considered here i.e. 512, 256 and 128 CPU cores. Allocated time period for running the application is provided as one hour. We have evaluated the execution time, performance impact in a private cluster deployed in our university research labs where Figure 3 and Figure 4 shows the comparable results using Torque and our proposed model. The cluster is having 22 Open Compute Windmill compute nodes, individually six 8 Intel Xeon E5-2660 CPU cores (2.20 GHz), 124 GB DDR4 with a 14 Gb/s Ethernet network. The shared filesystem is a NFS v4.2, with high performance of 10 Gb/s. The cluster is running on CentOS 7.0 and managed by Torque.

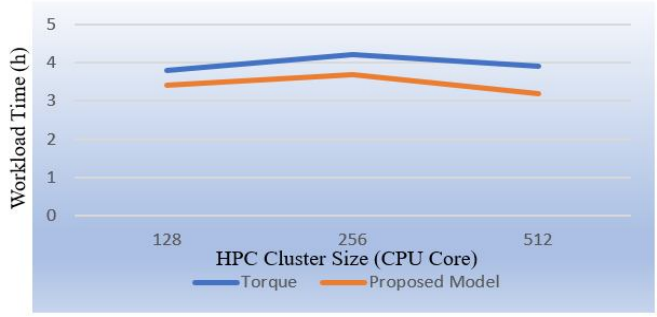

Figure 3: Average Queue Make span

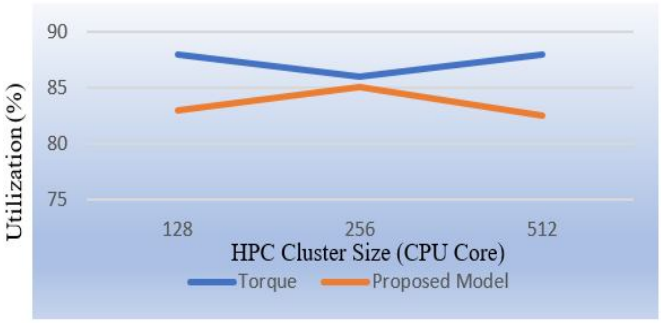

Figure 4: Average Resource Utilization

4.1 Use Case

We ran all two HPC application and one non-HPC application using the core 32, 8 and 6 for each. Every application has been executed isolated to form a baseline, cgroups based static sharing. Later, we compare the results using our proposed model using a parallel freezer and a random freezer. Using parallel freezer, it does not allow the second application till first one is using all resource (here it is CPU core) where random freezer only stopping second application for a duration of time period. We have defined the priorities for the application so that application with higher priority will always get preference for the resource consumption and application with lower priority will wait. Using Torque logs, we have collected the metrics data for each use case that is going to have total CPU usage, execution time for individual submitted job and all jobs. Table 1 is showing results related to the execution time of HPC application that are impacted by the different configuration of resource controllers based on the number of CPU cores i.e. 32, 6 and 8 in count. We can see the execution time results are much better with our proposed architecture implementation in comparison to parallel and Random freezer.

Table 1: Total execution time impact

Cores	Cgroups			Parallel Freezer	Random Freezer	Proposed Model
	95-05	80-20	05-95			
8	1.04	1.03	1.34	1.02	1.11	1.07
16	1.12	1.13	1.45	1.08	1.23	1.11
32	1.33	1.29	1.74	1.08	1.21	1.26

4.2 Results

Using our proposed model considered use case results have been captured and is showing impacted execution time for two HPC based applications using different resource controllers. The static rules provide better results for higher priority applications but poor for the lower priority ones. Our proposed model provides comparable good results for both lower as well as higher even consistent with the variation of core usage. After getting these results, we also checked for performance and found it comparable with Mesos with different size of clusters. Whenever we decrease the count of cores in terms of 512 to 256 and 256 to 128, via the job count and cores for each job decreased with a minor variation of resource utilization. We observe that our model gains a shorter make span than Torque. As Torque, does not come with user-driven scheduler to control the process scheduling by operating system, DevOps usually go with most common user satisfier. In contrast, our model facilitating a trade-off between performance and computing resource utilization, we user can actively monitor, profile, control the resource usages during variations in allocated resources or workloads.

Table 2 shows the comparison of Make Span, Core utilization and job throughput between the Torque usage and with our proposed model architecture.

Table 2: Results using Torque and proposed model

Queue	Make Span (sec)	Core Utilization (%)	Job throughput (@Job/Time [sec])
Torque	11954	60	4.2 * 10-4
Proposed Model	10260	75	1.2 * 10-3

4.3 Discussion

The testbeds results highlight the benefits of our proposed model using existing resource managers. With resource utilization in a consistent manner, the provided model is improving the dynamic resource allocation with comparable performance and reduced overheads in terms of deadlines, task execution waiting clock times. The DevOps should strictly define the rules for isolation as clubbing multiple resource managers in a cluster is not a good idea. Here the area related to fault tolerance, system crash, customer profiling [9] should be explored as variation in HPC cluster at runtime can generate disturbance. For profiling, any tool can be used wisely, and metrics data to handle the big data and its traits in public sector [12].

5. CONCLUSION

With the expansion of agility to scale infrastructure on demand and run complex application on them have introduced several hurdles in path of monitoring, profiling the computing resources. As everything is now revolving around data, HPC's premises offers promising

supercomputing functionalities and capabilities. In this paper, we have addressed the problem of dynamic resource allocation via proposing a model based on execution pattern which also helps to determine the free resources to run non-HPC applications. In future, we can also address the persistency of metrics data that is provided to users for duration of time. As life span grows, data is going to be huge and needs to be handled.

ACKNOWLEDGEMENT

A special vote of thanks to the Koneru Lakshmaiah Education Foundation for supporting and facilitating me required infrastructure and my guide as well as staff members who have helped me to complete this research work.

REFERENCES

1. S. Chen, B. Mulgrew, and P. M. Grant. **High Performance Computing (HPC) on AWS**. [http://aws.amazon.com/hpc-applications.]
2. **Torque Resource Manager overview and its advantages**. [https://adaptivecomputing.com/cherry-services/torque-resource-manager/]
3. **Slurm Overview**, its features and how to use it. [https://slurm.schedmd.com/overview.html]
4. **Mesos Overview**, its advantages and how it works. [http://mesos.apache.org/documentation/latest/]
5. Z. Fang, X. H. Sun, Y. Chen, S. Byna, **Core aware Memory Access Scheduling Schemes**. In IPDPS23,2009.
6. A Souza, M Rezaei, E Laure, J Tordsson, "**Hybrid Resource Management for HPC and Data Intensive Workloads**",2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID) https://doi.org/10.1109/CCGRID.2019.00054
7. R. M. Tomasul o. **An efficient algorithm for exploiting multiple arithmetic units**. In IBM Journal of Research and Development, Volume 11, Number 1, Page 25, 1967. https://doi.org/10.1147/rd.111.0025
8. D. T. Wang. **Modern DRAM Memory Systems Performance Analysis and a High Performance, Power Constrained DRAM Scheduling Algorithm**. Ph. D. Dissertation, Dept. Of ECE, University ofMaryland, 2005.
9. Wang, Gunawan & Maulana, Lazuardi & Leonardi, Nico & Kaburuan, Emil. (2020). **The Use of Big Data in Supporting Customer Profiling**. International Journal of Advanced Trends in Computer Science and Engineering.9.1128-1133.10.30534/ijatcse/2020/35922020.
10. K. Jackson et al. **Performance Analysis of High-Performance Computing Applications on the Amazon Web Services Cloud.** In CloudCom'10, 2010. https://doi.org/10.1109/CloudCom.2010.69
11. Kim, H., El-Khamra, Y., Jha, S., Parashar, M.: **Exploring application and infrastructure adaptation on hybrid grid-cloud infrastructure**. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, June 21-25, 2010, pp. 402–412 (2010). https://doi.org/10.1145/1851476.1851536
12. Wook, Muslihah. (2020). **Big Data Analytics Application Model Based on Data Quality Dimensions and Big Data Traits in Public Sector**. International Journal of Advanced Trends in Computer Science and Engineering. 9. 1247-1256. 10.30534/ijatcse/2020/53922020.
13. S.P. Bingulac, "**On the Compatibility of Adaptive Controllers**," Proc. Fourth Ann. Allerton Conf. Circuits and Systems Theory, pp. 8-16, 1994. (Conference proceedings).
14. H. Goto, Y. Hasegawa, and M. Tanaka, "**Efficient Scheduling Focusing on the Duality of MPL Representation**," Proc. IEEE Symp. Computational Intelligence in Scheduling (SCIS '07), pp. 57-64, Apr. 2007, doi:10.1109/SCIS.2007.367670. (Conference proceedings)
15. Nie, L., Xu, Z.: **An adaptive scheduling mechanism for elastic grid computing. In: International Conference on Semantics, Knowledge and Grid**, pp. 184–191 (2009).
16. Feitelson D. G.: **Parallel workloads** archive (PWA), February 2012
17. [http://www.cs.huji.ac.il/labs/parallel/workload/]
18. Feitelson D. G., Weil A. M.: **Utilization and predictability in scheduling the IBM SP2 with backfilling**. [in:] 12th International Parallel Processing Symposium, pages 542–546. IEEE, 1998
19. Kleban S. D., Clearwater S. H.: **Fair share on high performance computing systems: What does fair really mean**? [in:] Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03), pp. 146–153. IEEE Computer Society, 2003. https://doi.org/10.1109/CCGRID.2003.1199363
20. Klus´a˘cek D., Rudov´a H., Baraglia R., Pasquali M., Capannini G.: Comparison of multi-criteria scheduling techniques. [in:] **Grid Computing Achievements and Prospects**, pp. 173–184. Springer, 2008
21. SubW., Jakob W., Quinte A., Stucky K.-U.: GORBA: **A global optimising resource broker embedded in a Grid resource management system**. [in:] International Conference on Parallel and Distributed Computing Systems, PDCS 2005, pp. 19–24. IASTED/ACTA Press, 2005
22. Xhafa F., Abraham A.: **Metaheuristics for Scheduling in Distributed Computing Environments, volume 146 of Studies in Computational Intelligence**. Springer, 2008. https://doi.org/10.1007/978-3-540-69277-5
23. Xhafa F., Abraham A.: **Computational models and heuristic methods for Grid scheduling problems.**

Future Generation Computer Systems, 26(4):608–621, 2010.

24. Xu M. Q.: Effective metacomputing using LSF multicluster. [in:] CCGRID '01: **Proceedings of the 1st International Symposium on Cluster Computing and the Grid**, pp. 100–105. IEEE, 2001.
25. Kurowski K., Oleksiak A., Piatek W., Weglarz J.: **Hierarchical scheduling strategies for parallel tasks and advance reservations in grids. Journal of Scheduling**, 11(14):1–20, 2011. C. D. Locke, "**Best-effort Decision-making for Real-time Scheduling**," Ph.D. dissertation, Pittsburgh, PA, USA, 1986, aAI8702895.
26. P. Li and B. Ravindran, "**Fast, Best-Effort Real-Time Scheduling Algorithms**," IEEE Trans. Comput., vol. 53, no. 9, pp. 1159–1175, 2004. https://doi.org/10.1109/TC.2004.61
27. N. Bansal and K. R. Pruhs, "**Server Scheduling to Balance Priorities, Fairness, and Average Quality of Service**," SIAM J. Comput., vol. 39, no. 7, pp. 3311–3335, 2010. https://doi.org/10.1137/090772228
28. S. Aldarmi and A. Burns, "**Dynamic value-density for scheduling realtime systems**," in Proceedings of the Euromicro Conference on RealTime Systems, 1999, pp. 270–277.