International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Heuristic Based Resource Allocation for Cloud Using Virtual Machines

Ch Pavani[1], G Prabhakar Raju[2]

[1]M.tech Student, Department of CSE, Anurag Group of Institutions, Hyderabad, India

[2]Associate Professor, Department of CSE, Anurag Group of Institutions, Hyderabad, India

Abstract: ***Cloud computing allows to estimate the scale of resources for business customers .We achieve this through Virtualization Technology. Virtualization can be provided significant benefits in data centers by enabling virtual machine to eliminate hotspot. Virtualization used in scenarios-load balancing, online maintenance and proactive fault, power management. In Existing System VM Monitors like Xen provide a mechanism for mapping VM to physical resources. This mapping hidden from Cloud Users.VM live migration technology makes it possible to change the mapping between VM and PM while applications are running. Proposed system presents the implementation of an automatic resource management system that achieves a balance between the two goals-Avoidance Overload, Green Computing. By this we avoid Overload and introduce the concept of skewness to measure the uneven utilization of a Server.***

Keywords: Resource Management, Cloud Computing, Green Computing, Virtualization.

1. Introduction

The elasticity and the lack of upfront capital investment offered by cloud computing is appealing to many businesses. There is a lot of discussion on the benefits and costs of the cloud model and on how to move legacy applications onto the cloud platform. Here we study a different problem: how can a cloud service provider best multiplex its virtual resources onto the physical hardware? This is important because much of the touted gains in the cloud model come from such multiplexing. Studies have found that servers in many existing data centers are often severely under-utilized due to over-provisioning for the peak demand [1] [2]. The cloud model is expected to make such practice unnecessary by offering automatic scale up and down in response to load variation. Besides reducing the hardware cost, it also saves on electricity which contributes to a significant portion of the operational expenses in large data centers.

Virtual machine monitors (VMMs) like Xen provide a mechanism for mapping virtual machines (VMs) to physical resources [3]. This mapping is largely hidden from the cloud

users. Users with the Amazon EC2 service [4], for example, do not know where their VM instances run. It is up to the cloud provider to make sure the underlying physical machines (PMs) have sufficient resources to meet their needs. VM live migration technology makes it possible to change the mapping between VMs and PMs while applications are running [5], [6]. However, a policy issue remains as how to decide the mapping adaptively so that the resource demands of VMs are met while the number of PMs used is minimized. This is challenging when the resource needs of VMs are heterogeneous due to the diverse set of applications they run and vary with time as the workloads grow and shrink.
We aim to achieve two goals in our algorithm:

- Overload avoidance: the capacity of a PM should be ufficient to satisfy the resource needs of all VMs running on it. Otherwise, the PM is overloaded and can lead to degraded performance of its VMs.
- Green computing: the number of PMs used should be minimized as long as they can still satisfy the needs of all VMs. Idle PMs can be turned off to save energy.

There is an inherent trade-off between the two goals in the face of changing resource needs of VMs. For overload avoidance, we should keep the utilization of PMs low to reduce the possibility of overload in case the resource needs of VMs increase later. For green computing, we should keep the utilization of PMs reasonably high to make efficient use of their energy.
In this paper, we present the design and implementation of an automated resource management system that achieves a good balance between the two goals. We make the following contributions:

- We develop a resource allocation system that can avoid overload in the system effectively while minimizing the number of servers used.
- We introduce the concept of “skewness” to measure the uneven utilization of a server. By minimizing skewness, we can improve the overall utilization of servers in the face of multi-dimensional resource constraints.
- We design a load prediction algorithm that can capture the future resource usages of applications accurately without looking inside the VMs. The algorithm can capture the rising trend of resource usage patterns and help reduce the placement churn significantly.

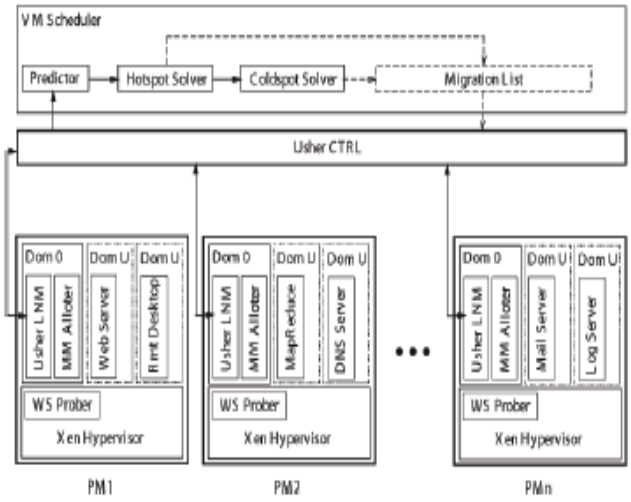

Figure 1: System Archtecture

The rest of the paper is organized as follows. Section 2 provides an overview of our system and Section 3 describes our algorithm to predict resource usage.

2. System Overview

The architecture of the system is presented in Figure 1. Each PM runs the Xen hypervisor (VMM) which supports a privileged domain 0 and one or more domain U [3]. Each VM in domain U encapsulates one or more applications such as Web server, remote desktop, DNS, Mail, Map/Reduce, etc. We assume all PMs share a backend storage.

The multiplexing of VMs to PMs is managed using the Usher framework [7]. The main logic of our system is implemented as a set of plug-ins to Usher. Each node runs an Usher local node manager (LNM) on domain 0 which collects the usage statistics of resources for each VM on that node. The CPU and network usage can be calculated by monitoring the scheduling events in Xen. The memory usage within a VM, however, is not visible to the hypervisor. One approach is to infer memory shortage of a VM by observing its swap activities [8]. Unfortunately, the guest OS is required to install a separate swap partition. Instead we implemented a working set prober (WS Prober) on each hypervisor to estimate the working set sizes of VMs running on it. We use the random page sampling technique as in the VMware ESX Server [9].

The statistics collected at each PM are forwarded to the Usher central controller (Usher CTRL) where our VM scheduler runs. The VM Scheduler is invoked periodically and receives from the LNM the resource demand history of VMs, the capacity and the load history of PMs, and the current layout of VMs on PMs.

The scheduler has several components. The predictor predicts the future resource demands of VMs and the future load of PMs based on past statistics. We compute the load of a PM by aggregating the resource usage of its VMs. The details of the load prediction algorithm will be described in the next section. The LNM at each node first attempts to satisfy the new demands locally by adjusting the resource allocation of VMs sharing the same VMM.

3. Predicting Future Resource Needs

We need to predict the future resource needs of VMs. As said earlier, our focus is on Internet applications. One solution is to look inside a VM for application level statistics, e.g., by parsing logs of pending requests. Doing so requires modification of the VM which may not always be possible. Instead, we make our prediction based on the past external behaviors of VMs. Our first attempt was to calculate an exponentially weighted moving average (EWMA) using a TCP-like scheme:

$$E(t) = \alpha * E(t-1) + (1-\alpha) * O(t), 0 \leq \alpha \leq 1$$

where $E(t)$ and $O(t)$ are the estimated and the observed load at time t, respectively. α reflects a tradeoff between stability and responsiveness.

We use the EWMA formula to predict the CPU load on the DNS server in our university. We measure the load every minute and predict the load in the next minute. Each dot in the figure is an observed value and the curve represents the predicted values. Visually, the curve cuts through the middle of the dots which indicates a fairly accurate prediction. This is also verified by the statistics in Table 1. The parameters in the parenthesis are the α values. W is the length of the measurement window (explained later). The "median" error is calculated as a percentage of the observed value: $|E(t) - O(t)|/O(t)$. The "higher" and "lower" error percentages are the percentages of predicted values that are higher or lower than the observed values, respectively. As we can see, the prediction is fairly accurate with roughly equal percentage of higher and lower values.

TABLE 1
Load prediction algorithms

	ewma(0.7) $W = 1$	fusd(-0.2, 0.7) $W = 1$	fusd(-0.2, 0.7) $W = 8$
median error	5.6%	9.4%	3.3%
high error	56%	77%	58%
low error	44%	23%	41%

On the other hand, when the observed resource usage is going down, we want to be conservative in reducing our estimation. Hence, we use two parameters, $\alpha_{\uparrow}$ and $\alpha_{\downarrow}$, to control how quickly $E(t)$ adapts to changes when $O(t)$ is increasing or decreasing, respectively. We call this the FUSD (Fast Up and Slow Down) algorithm. Now the predicted values are higher than the observed ones most of the time: 77% according to Table 1.

$$\begin{aligned} E(t) &= -|\alpha| * E(t-1) + (1+|\alpha|) * O(t) \\ &= O(t) + |\alpha| * (O(t) - E(t-1)) \end{aligned}$$

The median error is increased to 9:4% because we trade accuracy for safety. It is still quite acceptable nevertheless. So far we take $O(t)$ as the last observed value. Most applications have their SLOs specified in terms of a certain percentiles of requests meeting a specific performance level. More generally, we keep a window of W recently observed values and take $O(t)$ as a high percentile of them.

4. The Skewness Algorithm

We introduce the concept of *skewness* to quantify the unevenness in the utilization of multiple resources on a server. Let *n* be the number of resources we consider and r_i be the utilization of the *i*-th resource. We define the resource skewness of a server *p* as

$$skewness(p) = \sqrt{\sum_{i=1}^{n}(\frac{r_i}{\overline{r}} - 1)^2}$$

where $\overline{r}$ is the average utilization of all resources for server *p*. In practice, not all types of resources are performance critical and hence we only need to consider bottleneck resources in the above calculation. By minimizing the *skewness*, we can combine different types of workloads nicely and improve the overall utilization of server resources. In the following, we describe the details of our algorithm. Analysis of the algorithm is presented in Section 1 in the complementary file.

4.1 Hot and cold spots

Our algorithm executes periodically to evaluate the resource allocation status based on the predicted future resource demands of VMs. We define a server as a *hot spot* if the utilization of any of its resources is above a *hot threshold*. This indicates that the server is overloaded and hence some VMs running on it should be migrated away. We define the *temperature* of a hot spot p as the square sum of its resource utilization beyond the hot threshold:

$$temperature(p) = \sum_{r \in R}(r - r_t)^2$$

4.2 Hot spot mitigation

We sort the list of hot spots in the system in descending temperature (i.e., we handle the hottest one first). Our goal is to eliminate all hot spots if possible. Otherwise, keep their temperature as low as possible. For each server *p*, we first decide which of its VMs should be migrated away. We sort its list of VMs based on the resulting temperature of the server if that VM is migrated away. We aim to migrate away the VM that can reduce the server's temperature the most. In case of ties, we select the VM whose removal can reduce the skewness of the server the most. For each VM in the list, we see if we can find a destination server to accommodate it. The server must not become a hot spot after accepting this VM. Among all such servers, we select one whose skewness can be reduced the most by accepting this VM. Note that this reduction can be negative which means we select the server whose skewness increases the least. If a destination server is found, we record the migration of the VM to that server and update the predicted load of related servers. Otherwise, we move on to the next VM in the list and try to find a destination server for it. As long as we can find a destination server for any of its VMs, we consider this run of the algorithm a success and then move on to the next hot spot. Note that each run of the algorithm migrates away at most one VM from the overloaded server.

4.3 Green computing

When the resource utilization of active servers is too low, some of them can be turned off to save energy. This is handled in our green computing algorithm. The challenge here is to reduce the number of active servers during low load without sacrificing performance either now or in the future. We need to avoid oscillation in the system.

Our green computing algorithm is invoked when the average utilizations of all resources on active servers are below the green computing threshold. We sort the list of cold spots in the system based on the ascending order of their memory size. Since we need to migrate away all its VMs before we can shut down an under-utilized server, we define the memory size of a cold spot as the aggregate memory size of all VMs running on it. Recall that our model assumes all VMs connect to a shared back-end storage. Hence, the cost of a VM live migration is determined mostly by its memory footprint. The Section 7 in the complementary file explains why the memory is a good measure in depth. We try to eliminate the cold spot with the lowest cost first.

For a cold spot *p*, we check if we can migrate all its VMs somewhere else. For each VM on *p*, we try to find a destination server to accommodate it. The resource utilizations of the server after accepting the VM must be below the *warm threshold*. While we can save energy by consolidating under-utilized servers, overdoing it may create hot spots in the future. The warm threshold is designed to prevent that. If multiple servers satisfy the above criterion, we prefer one that is not a current cold spot. This is because increasing load on a cold spot reduces the likelihood that it can be eliminated. However, we will accept a cold spot as the destination server if necessary. All things being equal, we select a destination server whose skewness can be reduced the most by accepting this VM. If we can find destination servers for all VMs on a cold spot, we record the sequence of migrations and update the predicted load of related servers. Otherwise, we do not migrate any of its VMs.

5. Simulations

We evaluate the performance of our algorithm using trace driven simulation. Note that our simulation uses the same code base for the algorithm as the real implementation in the experiments. This ensures the fidelity of our simulation results. Traces are per-minute server resource utilization, such as CPU rate, memory usage, and network traffic statistics, collected using tools like "perfmon" (Windows), the "/proc" file system (Linux), "pmstat/vmstat/netstat" commands (Solaris), etc.. The raw traces are pre-processed into "Usher" format so that the simulator can read them. We collected the traces from a variety of sources:

- Web InfoMall: the largest online Web archive in China (i.e., the counterpart of Internet Archive in the US) with more than three billion archived Web pages.
- RealCourse: the largest online distance learning system in China with servers distributed across 13 major cities.
- AmazingStore: the largest P2P storage system in China.

We also collected traces from servers and desktop computers in our university including one of our mail servers, the central DNS server, and desktops in our department.We post-processed the traces based on days collected and use random sampling and linear combination of the data sets to
generate the workloads needed. All simulation in this section uses the real trace workload unless otherwise specified.

5.1 Effect of thresholds on APMs

We first evaluate the effect of the various thresholds used in our algorithm. We simulate a system with 100 PMs and 1000 VMs (selected randomly from the trace). We use random VM to PM mapping in the initial layout. The scheduler is invoked once per minute. The bottom part of Figure 2 show the daily load variation in the system. The x-axis is the time of the day starting at
Fig. 2. Impact of thresholds on the number of APMs

8am. The y-axis is overloaded with two meanings: the percentage of the load or the percentage of APMs (i.e., Active PMs) in the system. Recall that a PM is *active* (i.e., an APM) if it has at least one VM running. As can be seen from the figure, the CPU load demonstrates diurnal patterns which decreases substantially after midnight. The memory consumption is fairly stable over the time. The network utilization stays very low.

5.2 Scalability of the algorithm

We evaluate the scalability of our algorithm by varying the number of VMs in the simulation between 200 and 1400. The ratio of VM to PM is 10:1. The average decision time of our algorithm increases with the system size. The speed of increase is between linear and quadratic. We break down the decision time into two parts: hot spot mitigation (marked as 'hot') and green computing (marked as 'cold'). We find that hot spot mitigation contributes more to the decision time. We also find that the decision time for the synthetic workload is higher than that for the real trace due to the large variation in the synthetic workload. With 140 PMs and 1400 VMs, the decision time is about 1.3 seconds for the synthetic workload and 0.2 second
for the real trace

6. Experiments

Our experiments are conducted using a group of 30 Dell PowerEdge blade servers with Intel E5620 CPU and 24GB of RAM. The servers run Xen-3.3 and Linux 2.6.18. We periodically read load statistics using the xenstat library (same as what xentop does). The servers are connected over a Gigabit ethernet to a group of four NFS storage servers where our VM Scheduler runs. We use the same default parameters as in the simulation.

6.1 Algorithm effectiveness

We evaluate the effectiveness of our algorithm in overload mitigation and green computing. We start with a small scale experiment consisting of three PMs and five VMs so that we can present the results for all servers in figure 3. Different shades are used for each VM. All VMs are configured with 128 MB of RAM. An Apache server runs on each VM. We use httperf to invoke CPU intensive PHP scripts on the Apache server. This allows us to subject the VMs to different degrees of CPU load by adjusting the client request rates. The utilization of other resources are kept low.

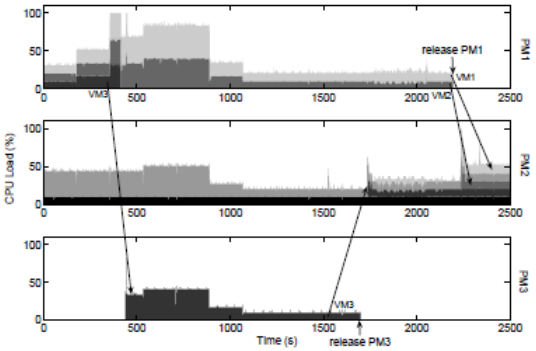

Figure 3: Algorithm effectiveness

6.2 Impact of live migration

One concern about the use of VM live migration is its impact on application performance. Previous studies have found this impact to be small [5]. We investigate this impact in our ownthe data on the 340 live migrations in our 30 server experiment above. We find that 139 of them are for hot spot mitigation. We focus on these migrations because that is when the potential impact on application performance is the most. Among the 139 migrations, we randomly pick 7 corresponding TPC-W sessions undergoing live migration. All these sessions run the "shopping mix" workload with 200 emulated browsers. As a target for comparison, we re-run the session with the same parameters but perform no migration and use the resulting performance as the baseline.

7. Related Work

7.1 Resource allocation at the Application Level

Automatic scaling of Web applications was previously studied in [14] [15] for data center environments. In MUSE [14], each server has replicas of all web applications running in the system. The dispatch algorithm in a frontend L7-switch makes sure requests are reasonably served while minimizing the number of under-utilized servers. Work [15] uses network flow algorithms to allocate the load of an application among its running instances. For connection oriented Internet services like Windows Live Messenger, work [10] presents an integrated approach for load dispatching and server provisioning. All works above do not use virtual machines and require the applications be structured in a multi-tier architecture with load balancing provided through an front-end dispatcher. In contrast, our work targets Amazon EC2-style environment where it places no restriction on what and how applications are constructed inside the VMs. A VM is treated like a blackbox. Resource management is done only at the granularity of whole VMs.

MapReduce [16] is another type of popular Cloud service where data locality is the key to its performance. Qunicy adopts min-cost flow model in task scheduling to maximize

data locality while keeping fairness among different jobs [17]. The "Delay Scheduling" algorithm trades execution time for data locality [18]. Work [19] assign dynamic priorities to jobs and users to facilitate resource allocation.

7.2 Resource allocation by live VM migration

VM live migration is a widely used technique for dynamic resource allocation in a virtualized environment [8] [12] [20]. Our work also belongs to this category. Sandpiper combines multi-dimensional load information into a single *Volume* metric [8]. It sorts the list of PMs based on their volumes and the VMs in each PM in their volume-to-size ratio (VSR). This unfortunately abstracts away critical information needed when making the migration decision. It then considers the PMs and the VMs in the pre-sorted order. We give a concrete example in Section 1 of the supplementary file where their algorithm selects the wrong VM to migrate away during overload and fails to mitigate the hot spot. We also compare our algorithm and theirs in real experiment. The results are analyzed in Section 5 of the supplementary file to show how they behave differently. In addition, their work has no support for green computing and differs from ours in many other aspects such as load prediction.

The HARMONY system applies virtualization technology across multiple resource layers [20]. It uses VM and data migration to mitigate hot spots not just on the servers, but also on network devices and the storage nodes as well. It introduces the *Extended Vector Product(EVP)* as an indicator of imbalance in resource utilization. Their load balancing algorithm is a variant of the Toyoda method [21] for multi-dimensional knapsack problem. Unlike our system, their system does not support green computing and load prediction is left as future work. In Section 6 of the supplementary file, we analyze the phenomenon that *V ectorDot* behaves differently compared with our work and point out the reason why our algorithm can utilize residual resources better.

Dynamic placement of virtual servers to minimize SLA violations is studied in [12]. They model it as a bin packing problem and use the well-known first-fit approximation algorithm to calculate the VM to PM layout periodically. That algorithm, however, is designed mostly for off-line use. It is likely to incur a large number of migrations when applied in on-line environment where the resource needs of VMs change dynamically.

7.3 Green Computing

Many efforts have been made to curtail energy consumption in data centers. Hardware based approaches include novel thermal design for lower cooling power, or adopting power-proportional and low-power hardware. Work uses Dynamic Voltage and Frequency Scaling(DVFS) to adjust CPU power according to its load. We do not use DVFS for green computing, as explained in the Section 7 in the complementary file. PowerNap resorts to new hardware technologies such as Solid State Disk(SSD) and Self-Refresh DRAM to implement rapid transition(less than 1ms) between full operation and low power state, so that it can "take a nap" in short idle intervals. When a server goes to sleep, Somniloquy notifies an embedded system residing on a special designed NIC to delegate the main operating system.

8. Conclusion

We have presented the design, implementation, and evaluation of a resource management system for cloud computing services. Our system multiplexes virtual to physical resources adaptively based on the changing demand. We use the skewness metric to combine VMs with different resource characteristics appropriately so that the capacities of servers are well utilized. Our algorithm achieves both overload avoidance and green computing for systems with multi-resource constraints.

References

[1] M. Armbrust *et al.*, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Tech. Rep., Feb 2009.
[2] L. Siegele, "Let it rise: A special report on corporate IT," in *The Economist*, Oct. 2008.
[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proc. of the ACM Symposium on Operating Systems Principles (SOSP'03)*, Oct. 2003.
[4] "Amazon elastic compute cloud (Amazon EC2), http://aws.amazon.com/ec2/."
[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. of the Symposium on Networked Systems Design and Implementation (NSDI'05)*, May 2005.
[6] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *Proc. of the USENIX Annual Technical Conference*, 2005.
[7] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker, "Usher: An extensible framework for managing clusters of virtual machines," in *Proc. of the Large Installation System Administration Conference (LISA'07)*, Nov. 2007.
[8] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. Of the Symposium on Networked Systems Design and Implementation (NSDI'07)*, Apr. 2007.
[9] C. A. Waldspurger, "Memory resource management in VMware ESX server," in *Proc. of the symposium on Operating systems design and implementation (OSDI'02)*, Aug. 2002.
[10] G. Chen, H. Wenbo, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proc. of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*, Apr. 2008.
[11] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proc. of the ACM European conference on Computer systems (EuroSys'09)*, 2009.

[12] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *Proc. of the IFIP/IEEE International Symposium on Integrated Network Management (IM'07)*, 2007.
[13] "TPC-W: Transaction processing performance council, http://www.tpc.org/tpcw/."
[14] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *Proc. Of the ACM Symposium on Operating System Principles (SOSP'01)*, Oct. 2001.
[15] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proc. Of the International World Wide Web Conference (WWW'07)*, May 2007.
[16] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. of the Symposium on Operating Systems Design and Implementation (OSDI'08)*, 2008.
[17] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in *Proc. of the ACM Symposium on Operating System Principles (SOSP'09)*, Oct. 2009.
[18] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proc. of the European conference on Computer systems (EuroSys'10)*, 2010.
[19] T. Sandholm and K. Lai, "Mapreduce optimization using regulated dynamic prioritization," in *Proc. of the international joint conference on Measurement and modeling of computer systems (SIGMETRICS'09)*, 2009.
[20] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers," in *Proc. of the ACM/IEEE conference on Supercomputing*, 2008.
[21] Y. Toyoda, "A simplified algorithm for obtaining approximate solutions to zero-one programming problems," *Management Science*, vol. 21, pp. 1417–1427, august 1975.

Author Profile

Ch Pavani received the B.Tech degree in Information Technology from JNTU HYDERABAD in 2012 and now pursuing M.Tech. degree in Computer science from CVSR COLLEGE OF ENGINEERING in JNTU Hyderabad.

G. Prabhakar Raju received the M. C. A degree from Osmania University and M. Tech degree in Computer Science and Engineering from JNTU University. He is an associate professor in the Department of Computer Science and Engineering, Anurag Group of Institutions.