ISSN: 2454-9940

# INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

E-Mail :
editor.ijasem@gmail.com
editor@ijasem.org

www.ijasem.org

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

https://doi.org/10.5281/zenodo.14351098

ISSN 2454-9940

www.ijasem.org

Vol 18, Issue 4, 2024

# Resource Over Commitment And User Centric Interference In Virtualized Environment

**K. BhavyaDeepika[1], Dr. M. SureshBabu [2]**

[1]PG Scholar, Department of CSE, Teegala Krishna Reddy Engineering College (Autonomous Institution), Medbowli, Meerpet, Saroornagar, Hyderabad

[2] Professor, Department of CSE, Teegala Krishna Reddy Engineering College (Autonomous Institution), Medbowli, Meerpet, Saroornagar, Hyderabad

## ABSTRACT

Modern distributed server applications are hosted on enterprise or cloud data centers that provide computing, storage, and networking capabilities to these applications. These applications are built using the implicit assumption that the underlying servers will be stable and normally available, barring for occasional faults. In many emerging scenarios, however, data centers and clouds only provide transient, rather than continuous, availability of their servers. Transiency in modern distributed systems arises in many contexts, such as green data centers powered using renewable intermittent sources, and cloud platforms that provide lower- cost transient servers which can be unilaterally revoked by the cloud operator. Transient computing resources are increasingly important, and existing fault tolerance and resource management techniques are inadequate for transient servers because applications typically assume continuous resource availability. This project presents research in distributed systems design that treats transiency as a first-class design principle. Combining transiency-specific fault-tolerance mechanisms with resource management policies to suit application characteristics and requirements, can yield significant cost and performance benefits. These mechanisms and policies have been implemented and prototyped as part of software systems, which allow a wide range of applications, such as interactive services and distributed data processing, to be deployed on transient servers, and can reduce cloud computing costs by up to 90%. This thesis makes contributions to four areas of computer systems research: transiency- specific fault-tolerance, resource allocation, abstractions, and resource reclamation. For reducing the impact of transient server revocations, two fault tolerance techniques that are tailored to transient server characteristics and application requirements are developed. For interactive applications, a derivative cloud platform that masks revocations by transparently moving application-state between servers of different types is constructed. Similarly, for distributed data processing applications, the use of application level

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
ISSN 2454-9940
www.ijasem.org
Vol 18, Issue 4, 2024
https://doi.org/10.5281/zenodo.14351098

periodic check pointing to reduce the performance impact of server revocations is investigated. For managing and reducing the risk of server revocations, the use of server portfolios that allow transient resource allocation to be tailored to application requirements is also investigated. Resource deflation generalizes revocation, and the deflation mechanisms and cluster-wide policies can yield both high cluster utilization and low application performance degradation.

**Keywords:**Resource deflation,User-centric Interference,SDLC,cloud platforms

## I. INTRODUCTION

Many enterprises and software systems rely in large part on cloud computing platforms for their computing needs. Today's cloud platforms enable customers to rent computing resources and deploy applications on them in an on demand manner. This utility-computing model offers numerous benefits, including pay-as-you-go pricing, the ability to quickly scale capacity when necessary, and low costs, due to their high degree of statistical multiplexing and massive economies of scale. To handle the growing number and diversity in applications, cloud platforms offer computing resources with a wide range of cost, availability, and performance characteristics. This project looks at one such type of computing resource, called transient servers. In contrast to traditional cloud servers whose availability can be assumed to be continuous, transient servers only offer intermittent and transient availability, and applications can have their access forcibly *revoked* by the resource provider. Running modern distributed applications on transient servers raises a slew of new challenges. Most applications are designed and built with the implicit assumption that its computing resources will continue to be available until relinquished. Transient server revocations can cause loss of application-state, which can result in application downtimes, degraded performance due to failure-recovery, and end-user dissatisfaction in general. While transient servers introduce many challenges for applications, they are also *significantly* cheaper compared to their non-revocable counterparts. For example, transient servers offered by large public cloud providers such as Amazon EC2's spot servers can be upto 50-90% cheaper compared to the traditional, non-revocable, "on-demand" servers. This project examines and addresses some of the challenges of running applications on cloud transient servers. These challenges are addressed by designing and building systems that introduce new mechanisms, policies, and abstractions—that together enable more effective use of transient servers for a wide range of applications.

## II. RELATED WORK

T. Wood et al. [4] give the black and grey box strategies with bg algorithm. Author uses xen hypervisor and finds with nucleus and monitoring engine, grey-box enables proactive decision making. While it has the limitation as, black-box is limited to reactive decision making and bg algorithm requires more number of migrations. A. Singh et al. [5] introduces the integrated server storage virtualization (vector dot algorithm) using configuration and performance manager. This scheme has a smaller amount of complication but its forecasting is not believable. Because of uneven distribution of remaining resource makes it hard to be fully utilized in the future. Zhen xiao [6] gives the strategy for dynamic resource allocation with skewness and load prediction algorithm. He uses xen hypervisor usher controller. The merits in his system are no overheads, high performance. It requires less number of migrations and residual resource is friendly to

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
ISSN 2454-9940
www.ijasem.org
Vol 18, Issue 4, 2024
https://doi.org/10.5281/zenodo.14351098

virtual machines. It improves the scheduling effectiveness. The demerit of the system is it is not cost effective. The benefit of shared space of cloud infrastructure explained by l. Qiang et al. [7] in which author proposed resource allocation strategy using feedback control theory, for suitable management of virtualized resources, which is based on virtual machine (vm). In this vm-based architecture all hardware resources are combined into common shared space in cloud computing infrastructure so that hosted application can right to use the required resources as per there need to meet service level objective (slos) of application. The adaptive manager use in this architecture is multi-input multi-output (mimo) resource manager, which consist of 3 controllers: cpu controller, memory controller and i/o controller, its goal is control multiple virtualized resources utilization to achieve slos of application by using control inputs per-vm cpu, memory and i/o allocation. Utility functions provide a natural and advantageous framework for achieving self-optimization in distributed autonomic computing systems explained by walsh et al. [8]. Author present a distributed architecture, implemented in a realistic prototype data center that demonstrates how utility functions can enable a collection of autonomic elements to continually optimize the use of computational resources in a dynamic, heterogeneous environment. The architecture consists of a two-level structure of autonomic elements that supports elasticity, modularity, and self-management. Each individual autonomic element manages application resource usage to optimize local service-level utility functions, and a global arbiter maps resources among application environments based on resource-level utility functions obtained from the managers of the applications. The utility function scheme is suitable for handling realistic, fluctuating web-based transactional workloads running on a linux cluster. Resource provision based on updated actual task executed explained by jiayin li et al. [9] which proposes an adaptive resource allocation algorithm for the cloud system with preempt able tasks in which algorithms adjust the resource provision adaptively based on the updated of the actual task executions. Author proposed adaptive list scheduling (als) and adaptive min-min scheduling (amms) algorithms and used for task scheduling which includes static task scheduling, for static resource allocation, is generated offline. Online adaptive method is use for re-evaluating the remaining static resource allotment repeatedly with predefined frequency. For every re-evaluation process, the schedulers are re-calculating the finish time of their respective submitted tasks, not the tasks that are assign to that cloud. So this method is suitable for static resource allocation. The dynamic resource allocation using distributed multiple criteria decisions in computing cloud explained by yazir y.o.et al. [10]. In it author contribution is tow-fold, first distributed architecture is adopted, in which resource management is separated into independent tasks, each of which is performed by autonomous node agents (na) in ac cycle of three activities: (i) vm placement, in it suitable physical machine (pm) is found which is capable of running given vm and then assigned vm to that pm, (ii) monitoring, in it total resources use by hosted vm are monitored by na, (iii) in vm selection, if local accommodation is not possible, a vm need to migrate at another pm and process loops back to into placement. Second using promethee method, node agent carry out configuration in parallel through multiple criteria decision analysis. This scheme is most suitable for large data centers as compared with centralized approaches. Nowadays distributed computing systems solves rising demand of computing and memory. In the distributed systems specifically resource allocation is one of the most important challenges while the clients have service level agreements (slas) and the whole profit in the system

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
https://doi.org/10.5281/zenodo.14351098
ISSN 2454-9940
www.ijasem.org
Vol 18, Issue 4, 2024

depends on how the system can meet these slas. This issue was solved by solved by goudarzi et al. [11] which optimizes the total profit gained from the multidimensional sla contracts for multi-tire application. In this scheme higher level of entire profit is provided by using force-directed resource assignment (fra) heuristic algorithm, in this case primary solution is based on provided solution for profit higher level problem. Then, distribution rates are set and local optimization step is use for improving resource sharing. Resource consolidation method is applied lastly to consolidate resources to determine the active (on) servers and further optimize the resource obligation. As concluding this method is suitable for improving resource sharing and to optimize the resource assignment. Use of steady state timing models, tafi. Et al. [12] presents information of cloud hpc resource arrangement. In which author proposed quantitative application dependent instrumentation scheme to inspect several important dimensions of a program's scalability. Sequential and parallel timing model with program instrumentations can reveal architecture exact deliverable performances that are difficult to measure otherwise. These models are introduces to connect s.k.sonkar et al., international journal of advanced trends in computer science and engineering, 4(4), july - august 2015, 48 - 51 50 several dimensions to time domain and application speed up model is use to tie these models in same equation. This provides ability to explore multiple dimension of program quantitatively to gain non--trivial insight. Authors use amazon ec2 as a target processing environment.

## III. SYSTEM ANALYSIS

Different cloud providers have employed different approaches for pricing transient servers. Google's transient servers, called preemptible instances, offer a fixed 80% discount but also have a maximum lifetime of 24 hours (with the possibility of earlier preemption). In contrast, Amazon's transient servers (which are called spot instances) offer a variable discount—the price of spot instances varies continuously based on market supply and demand for each server type (Figure 2.2). Spot instances are typically 0.1–0.5× the cost of non-revocable on-demand instances.

Since transient servers are surplus idle machines, the resources available in the transient server pool fluctuate continuously depending on the supply and demandof on-demand servers. Thus, whether a certain transient server is available depends on current market conditions. A combination of server-type (such as large/small), geographical region, and availability zone (data center failure domains within aregion), define a separate market of transient servers. The price and/or availability characteristics of individual markets can differ, as see, which shows EC2 spot prices. In this example, the m3.medium in availability zone a has the most stable prices, g2.2xlarge in the same availability zone has a lower average price but high variance, and the m3.medium in availability zone b has higher price.

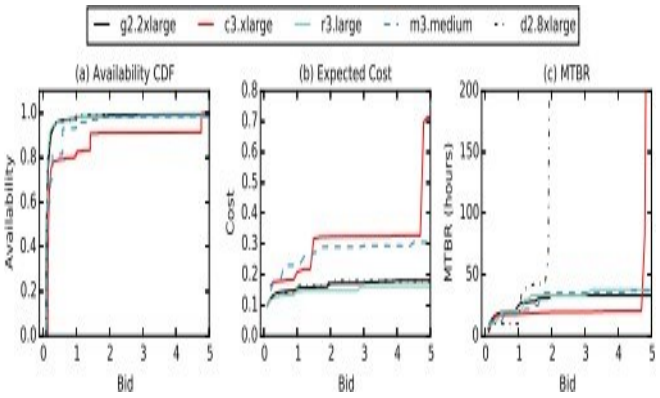



**Figure 2.4: The effect of bidding on availability, expected cost, and MTBR for selected instance types.**

Bids and the expected costs are normalized to a factor of thecorresponding on-demand price.high variance, and the

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
ISSN 2454-9940
www.ijasem.org
Vol 18, Issue 4, 2024
https://doi.org/10.5281/zenodo.14351098

m3.medium in availability zone b has higher price. The g2.2xlarge price spikes are not correlated with the other two servers. The example shows that larger servers may occasionally be more heavily discounted than smaller servers, and that identical servers in two availabilityzones may also be priced differently. The supply and demand of different server types across different regions may not always be correlated, and this is reflected in the general lack of correlation intheir spot prices (Figure 2.3).Bidding for EC2 spot instances. Amazon EC2 spot prices are determined by continuous sealed-bid second-price auction. Users place a single, fixed bid, which represents the maximum hourly price that they are willing to pay. The market price is based on all the bids and the available supply. Importantly, all users pay the same market price, which may be lower than the bid. The price of a spot instances in EC2 thus fluctuates continuously in real-time based on market demand and supply. If the spot price rises above a user's maximum bid price due to increased market demand, EC2 revokes the spot server from the user after providing a two minute warning (and presumably allocates it to a higher paying user).

## IV. IMPLEMENTATION

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

Software Development Paradigm:

The software development paradigm helps developer to select a strategy to develop the software. A software development paradigm has its own set of tools, methods and procedures, which are expressed clearly and defines software development life cycle. A software development paradigms or process models are defined as follows:

Spiral Model

Spiral model is a combination of both, iterative model and one of the SDLC model. It can be seen as if you choose one SDLC model and combine it with cyclic process (iterative model).

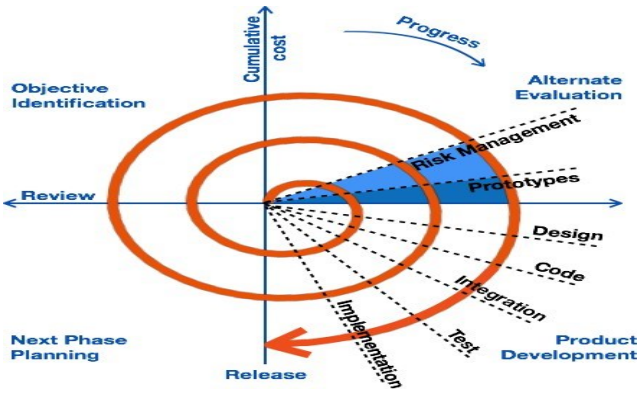



**Fig.5.1 Spiral Model**

This model considers risk, which often goes un-noticed by most other models. The model starts with determining objectives and constraints of the software at the start of one iteration. Next phase is of prototyping the software. This includes risk analysis. Then one standard SDLC model is used to build the software. In the fourth phase of the plan of next iteration is prepared.

SDLC Activities

SDLC provides a series of steps to be followed to design and develop a software product efficiently.

SDLC framework includes the following steps:

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
https://doi.org/10.5281/zenodo.14351098
ISSN 2454-9940
www.ijasem.org
Vol 18, Issue 4, 2024

**Fig.5.2 SDLC Communication**

This is the first step where the user initiates the request for a desired software product. He contacts the service provider and tries to negotiate the terms. He submits his request to the service providing organization in writing.

Requirement Gathering

This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given –

- studying the existing or obsolete system and software,
- conducting interviews of users and developers,
- referring to the database or
- Collecting answers from the questionnaires.

## V. RESULTS AND DISCUSSION

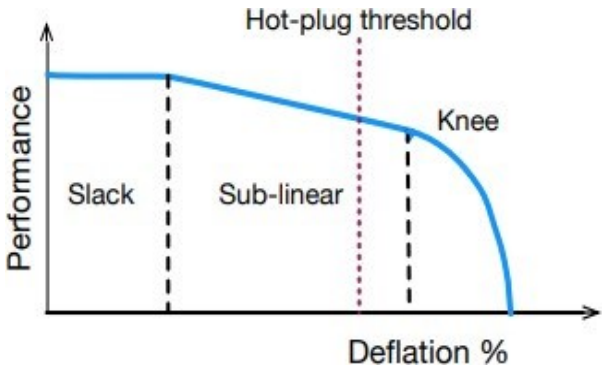
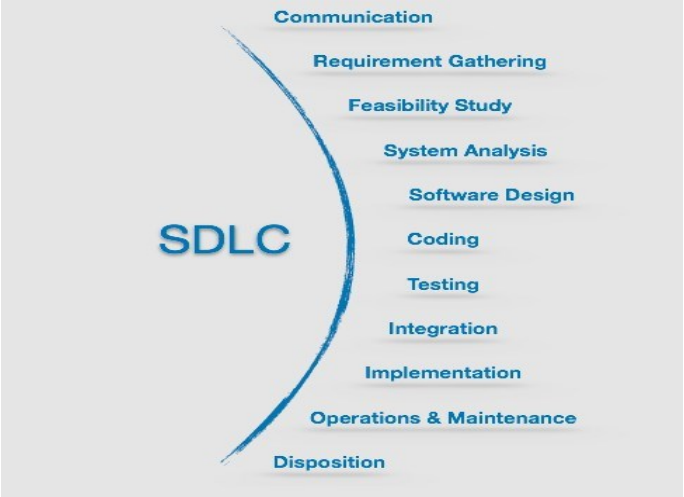



**Fig 1 : A representative deflation utility curve**



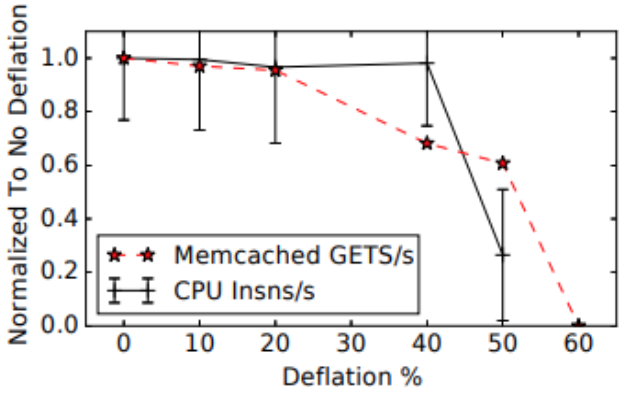
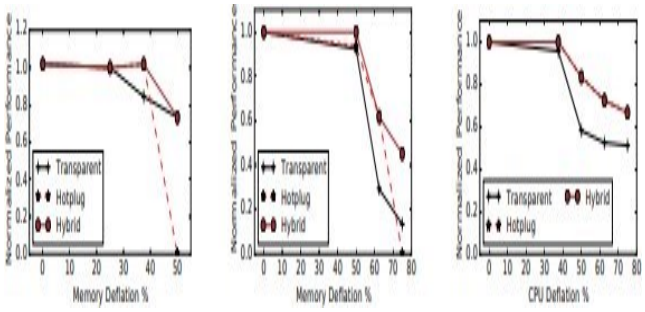




**Fig 2 : memcached performance shows high correlation with cpu counters (pearson correlation=0.72). Drop in counters predicts the knee at50% deflation.**




**(a) Memcached memory deflation(b) Kernel-compile memory deflation(c)kernel-compile cpu deflation**

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
https://doi.org/10.5281/zenodo.14351098
ISSN 2454-9940
www.ijasem.org
Vol 18, Issue 4, 2024

(a) Memcached (b) JVM (SpecJBB)

**Figure 3 : Deflation-aware application performance**

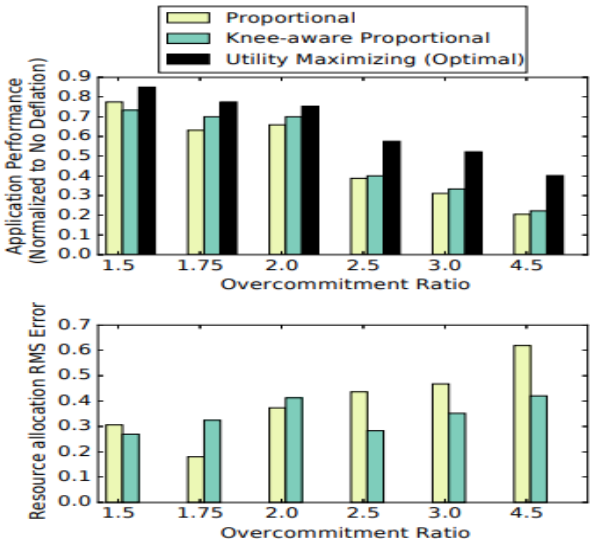
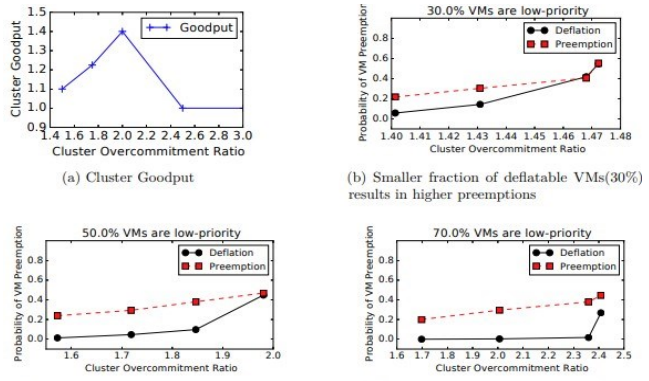



**Fig 4: Compared to optimal utility maximization, performance of VMs with the knee-aware proportional deflation is within 10%-50% of the optimal.**



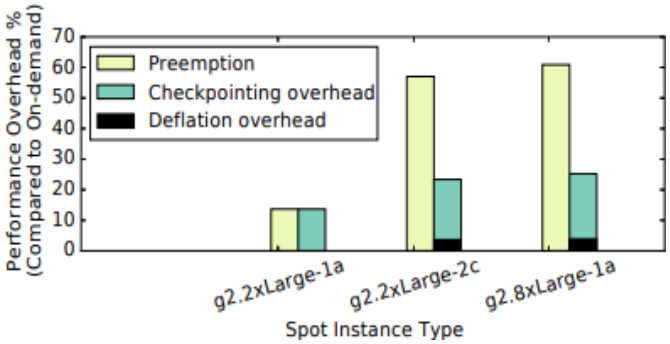

(a) Cluster Goodput

(b) Smaller fraction of deflatable VMs(30%) results in higher preemptions

(c) Deflation results in lower preemption probability

(d) Preemptions virtually eliminated if a high fraction of VMs are deflatable.

**Fig 5: Server overcommitment and preemption probabilities (right axis) with different VM placement policies.**



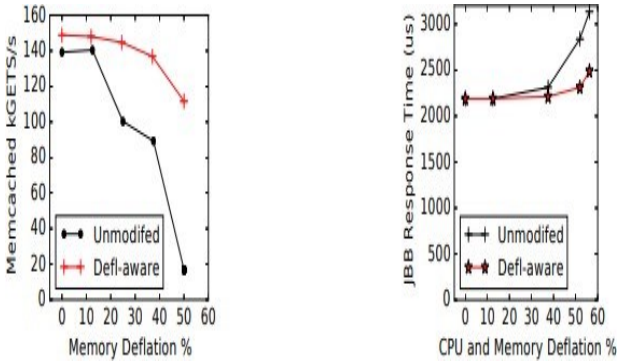




**Fig 6: Clusters and Premptions**

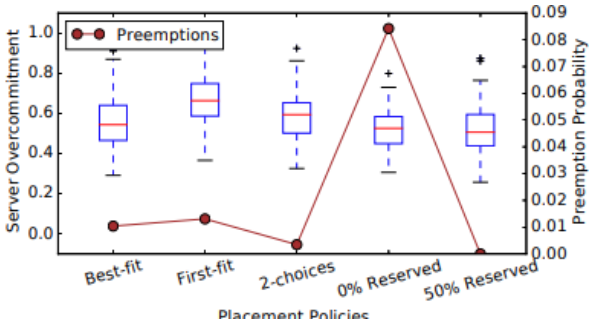



**Fig 7: Performance overhead of fault-tolerance when using preemption and deflation**

## VI. CONCLUSION

This dissertation presents a comprehensive exploration of transient resource availability in cloud computing. By addressing the challenges posed by transient servers with innovative fault-tolerance techniques and resource management policies, we have demonstrated the potential for significant cost savings and performance improvements. Resource over-commitment and user-centric interference are critical aspects to consider in a virtualized environment. Both involve managing resources efficiently and ensuring a high-quality user experience. Resource over-commitment refers to the practice of allocating more virtual resources to virtual machines (VMs) than the physical resources available on the host. This can be done for CPU, memory, and storage.

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN 2454-9940

www.ijasem.org

Vol 18, Issue 4, 2024

https://doi.org/10.5281/zenodo.14351098

Resource over commitment allows better utilization of physical resources, as not all VMs will use their maximum allocated resources simultaneously and reduces the need for additional hardware, leading to cost savings. This provides flexibility in managing workloads and scaling up services dynamically. The systems developed and described herein represent a major advancement in the practical use of transient resources, providing a robust framework for a wide range of applications to benefit from this exciting and important resource allocation model

## REFERENCES

[1] Alibaba cloud. http://www.alibabacloud.com.
[2] Amazon ec2 instance offerings. https://aws.amazon.com/ec2.
[3] Amazon web services - cloud computing services. https://aws.amazon.com/.
[4] Google cloud computing. http://cloud.google.com.
[5] Ibm cloud. http://www.ibm.com/cloud.
[6] Joyent public cloud. http://www.joyent.com.
[7] Memcached. https://memcached.org/.
[8] Microsoft azure cloud computing platform and services. http://azure. microsoft.com.
[9] QEMU Microcheckpointing. http://wiki.qemu.org/Features/ MicroCheckpointing.
[10] SPECjbb2005. https://www.spec.org/jbb2005/.
[11] TPC-W Benchmark. http://jmob.ow2.org/tpcw.html.
[12] Google's Green PPAs: What, How, and Why. http://www.google.com/green/pdfs/renewable-energy.pdf, April 2011.
[13] Heroku. http://www.heroku.com, May 1st 2014.
[14] PiCloud. http://www.multyvac.com, May 1st 2014.
[15] RightScale. http://rightscale.com, May 1st 2014.
[16] Single Root I/O Virtualization. https://www.pcisig.com/specifications/iov/single_root/,May 1st 2014.
[17] Amazon EC2 Spot Instances. https://aws.amazon.com/ec2/spot/, Septem-ber 24th 2015.
[18] Amazon Elastic Map Reduce for Spark. https://aws.amazon.com/ elasticmapreduce/details/spark/, June 2015.
[19] Docker. https://www.docker.com/, June 2015.
[20] Ec2 spot bid advisor. https://aws.amazon.com/ec2/spot/bid-advisor/, September 2015.
[21] Ec2 Spot Blocks, October 2015. https://aws.amazon.com/about-aws/whats-new/2015/10/introducing-amazon-ec2-spot-instances-for-specific- duration- workloads/.
[22] Ec2 spot-fleet. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot- fleet.html, September 2015.
[23] Eucalyptus workload traces. https://www.cs.ucsb.edu/~rich/workload/, 2015.
[24] Google preemptible instances. https://cloud.google.com/compute/docs/instances/preemptible, September 24th 2015.
[25] Livejournal Social Network Dataset. https://snap.stanford.edu/data/soc- LiveJournal1.html, June 2015.
[26] Lxc. https://linuxcontainers.org/, June 2015.
[27] Openstack. https://www.openstack.org, June 2015.
[28] Transaction Processing Performance Council - Benchmark H. http://www.tpc. org/tpch/, June 2015.
[29] Cloudstack. https://cloudstack.apache.org/, March 2016.
[30] Docker Swarm. https://www.docker.com/products/docker-swarm, March2016.
[31] Hadoop Recovery. https://twiki.grid.iu.edu/bin/view/Storage/HadoopRecovery, March 2016.
[32] Kubernetes. https://kubernetes.io, June 2016.
[33] Lxd. https://linuxcontainers.org/lxd/, January 2016.
[34] Mpich: High performance portable mpi. https://www.open-mpi.org/, 2016.
[35] Openmpi checkpointing. https://www.open-mpi.org/faq/?category=ft, 2016.
[36] Risk-return trade-off. http://cvxopt.org/examples/book/portfolio.html, 2016.
[37] VMware ESX hypervisor. https://www.vmware.com/products/vsphere-hypervisor, March 2016.
[38] VMware vCenter. https://www.vmware.com/products/vcenter-server, March 2016.