PERFORMANCE ANALYSIS BASED RESOURCE ALLOCATION, DAFT AND PAYMENT MINIMIZATION FOR CLOUD COMPUTING

R.S.Prabakaran,
PG Scholar,
Sri Ramakrishna Engineering coll,
Coimbatore.

Dr.P.Mathiyalagan,
Associate Professor.
Sri Ramakrishna Engineering College,
Coimbatore.

ABSTRACT

To maximize utilization and minimize total cost of the cloud computing infrastructure and running applications, resources need to be managed properly and virtual machines shall allocate proper host nodes . In this work, we propose performance analysis based resource allocation scheme for the efficient allocation of virtual machines on the cloud infrastructure. Fault tolerant service is an essential part of Service Level Objectives (SLOs) in clouds. To achieve high level of cloud serviceability and to meet high level of cloud SLOs. DAFT includes: (i) analyzing the mathematical relationship between different failure rates and different fault tolerance strategies, which are checkpointing fault tolerance strategy (ii) Building a dynamic adaptive checkpointing fault tolerance model combine together to maximize the serviceability and meet the SLOs(iii) Evaluating the dynamic adaptive fault tolerance strategy under various conditions in large-scale cloud data centers . It efficiently and effectively achieves a trade-off for fault tolerance objectives in cloud computing environment and (iv) Balances frequent check pointing strategy and infrequent check pointing strategy and provides dynamic adaptive check pointing strategy. We extend our work to scheduling based on minimizing makespan, cost . Our experimental results shows that our work more efficient for scheduling and resource allocation and improving the resource utilization.

Keywords: Cloud Computing,Resource allocation,Performance analysis,VMs,Fault tolerance,Checkpointing,Cost-efficient scheduling.

ISSN: 2278 – 7798

1. INTRODUCTION:

Cloud Computing[1] is a model for enabling convenient, on-demand network access to a shared pool of configurable and reliable computing resources (e.g., networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal consumer management effort or service provider interaction. Cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a metered service over a network (typically the Internet). Cloud computing provides computation, software, data access, and storage resources without requiring cloud users to know the location and other details of the computing infrastructure. Cloud computing is transforming business by offering new options for businesses to increase efficiencies while reducing costs. These problems include:

- High operational costs: typically associated with implementing and managing desktop and server infrastructures
- Low system utilization: often associated with non-virtualized server workloads in enterprise environments
- Inconsistent availability: due to the high cost of providing hardware redundancy
- Poor agility: which makes it difficult for businesses to meet evolving market demands

The efficient resource allocation optimization problem is conducted by sub problems. The proposed cloud resource allocation optimization algorithm is achieved through an iterative algorithm. The proposed efficient resource allocation for optimizing the objectives of cloud users, IaaS provider and SaaS provider in cloud computing is conducted by sub problems. In each iteration, the cloud users compute the unique optimal payment to SaaS provider under the deadline constraint to maximize the cloud user's satisfaction. The SaaS provider never sells a SaaS service for less than the cost paid to the IaaS providers for supporting VMs. So SaaS provider adjusts its cloud resource demand for running VMs under budget constraint. Different IaaS providers compute optimal resource allocation for maximizing the revenue of their own.

2. RELATED WORKS:

J. Wong et al.[3] proposed a Static resource allocation based on peak demand is not cost-effective because of poor resource utilization during off-peak periods.. Resource provisioning for cloud computing, an important issue is how resources may be allocated to an application mix such that the service level agreements (SLAs) of all applications are met Heuristic algorithm that determines a resource allocation strategy (SA or DA) that results in the smallest number of servers required to meet the SLA of both classes; Comparative evaluation of FCFS, head-of-the-line priority (HOL) and a new scheduling discipline called probability dependent priority (PDP). Scott et al[5] proposed a Finding the failure rate of a system is a crucial step in high performance computing systems analysis. Fault tolerant mechanism, called checkpoint/ restart technique, was introduced. Incremental checkpoint model can reduce the waste time more than it is reduced by the full checkpoint model. Singh et al. presented a slot-based provisioning model on grids to provide scheduling according to the availability and cost of resources.

3. PROPOSED SYSTEM:

Task-scheduling algorithm [9][10]for running large programs in the cloud. Most conventional task scheduling algorithms do not consider monetary costs, and so they cannot be directly applied in a cloud setting. Existing system algorithm computes scheduling plans that produce make span as good as the best known algorithm of while significantly reducing monetary costs. We use the concept of Pareto dominance to devise a cost-efficient scheduling algorithm to process multiple tasks in the cloud setting.

Pareto optimality

Incorporating monetary cost into task scheduling adds a layer of complexity. Since we cannot directly compare different scheduling plans with competing objectives, we will use the concept of Pareto dominance to select VMs and carry out comparisons.

Pareto optimal scheduling heuristic (POSH)

It describes a heuristic to dispatch tasks in a DAG to the cost-conscious VMs based on Pareto dominance, and we call it Pareto Optimal Scheduling Heuristic (POSH). POSH is an extension of the Heterogeneous Earliest Finish Time (HEFT) heuristic. Developed for scheduling tasks on heterogeneous dedicated multiprocessing systems, HEFT is better than other scheduling heuristics. HEFT assigns a priority to each task and then maps the task with the highest priority to the VM that minimizes the earliest finish time. POSH uses both the running time and the monetary cost to modify the last step to map the task with the highest priority to the most cost-efficient VM based on Pareto dominance.

ADVANTAGES OF PROPOSED SYSTEM

- The algorithm can substantially reduce monetary costs while producing make span as good as the best known task-scheduling algorithm can provide.
- To find a schedule of these tasks so that the monetary costs and the make span are both minimized.

ISSN: 2278 – 7798

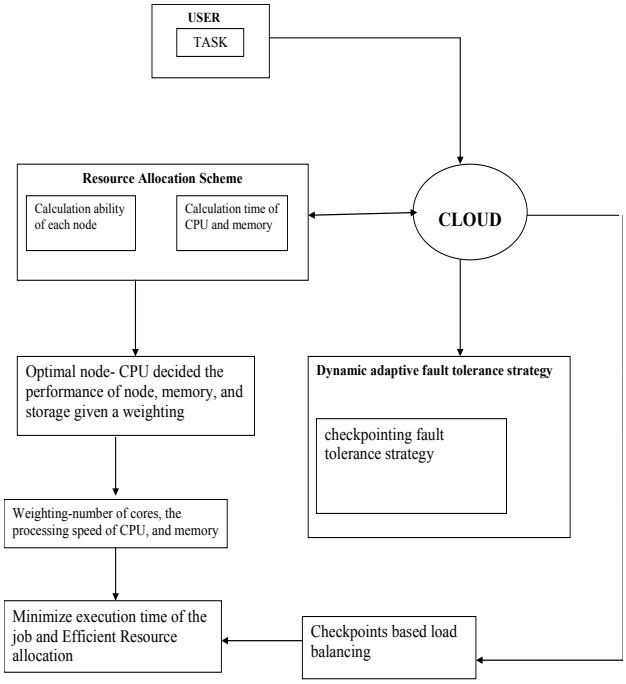

Fig no: Architecture diagram

4. SYSTEM IMPLEMENTATION:

4.1 Allocation of Virtual Machines Obtaining the Inverse Matrix

$$L = F\lceil lowerMat(A)\rceil (1)$$
$$U = F\lceil upperMat(A)\rceil (2)$$
$$L_{inv} = F\lceil inverseMat(L)\rceil (3)$$
$$U_{inv} = F\lceil inverseMat(U)\rceil (4)$$

The inverse matrix of [A] is also obtained by the multiplication of the inverse matrix of both [L] and [U]. The inverse matrix equation of [A] is as follows:

$A_{inv} = U_{inv} \cdot L_{inv}$

NiCore Value = Ni · core/coremax…. (5)

NiCpu Value = Ni · cpu/cpumax …(6)

NiMem Value = Ni · mem/memmax….(7)

Ni = (corevalue ×α) +(cpuvalue ×β)+(memvalue ×γ)…..(8)

4.2 Resource allocation algorithm

Resource allocation is the process of assigning available resources to the needed cloud applications. Cloud resources consist of physical and virtual resources. The user request for virtualized resources is described through a set of parameters detailing the processing CPU, memory, disk, and so on.

For each i ∈ Node(Core,CPU,Mem)

Starttime←Times();

Memvalue←InvertMatrix(Ni);

Finishtime←Times();

CPUvalue←Finishtime − Starttime;

N1←(corevalue ×0.2)+(cpuvalue ×0.5)+(memvalue ×0.3);

DB.add(Ni);

end for

Node performance analysis algorithm

for each i ∈ N.size()
if !available(Ni , requestResource)
availableNodeList.add(Ni);
end if
end for
Sort (availableNodeList);
while j ≥ availableNodeList.size()
if VM = empty
creatVM(VM);
end if
if success(Nj ←VM)
vmtable.add(j,VM);
end if
j++;
end while

Virtual machine scheduling algorithm

After getting the proper host, the scheduler will return the host number to the virtual machine manager for placement of virtual machine on that host. Then the virtual machine manager has all information about the virtual machine and its location. It will send a service activation message to the client/user. After that, the client/user can access the service for the duration specified. And when the resources and the data are ready, this task's execution begins.

4.3 Dynamic adaptive check pointing strategy

In order to optimize the checkpointing interval ΔT, the optimization checkpointing density function $\rho(t)$ can be obtained by minimizing the failure distribution value $E(T_{cyc})$, as shown

$$\rho(t) = \left(\frac{1}{T_{chk}} \cdot \frac{f(t)}{(1-F(t))}\right)^{0.5} \ldots.(9)$$
$$\Delta T = \left(T_{chk} \cdot \frac{(1-F(t))}{f(t)}\right)^{0.5} \ldots(10)$$

If the failure density function is f (t), the failure distribution function is F(t), the overhead of each check pointing overhead is T_{chk}, then the check pointing density function $\rho(t)$ and the check pointing interval ΔT can be calculated.

$$T_{free} = T_{chk} . \int_{t1}^{tk} p(\tau) d\tau \ldots(11)$$

Where,

T_{free} = The check pointing overhead during the longest failure-free time interval of the consecutive checkpoints

Tchk =The time interval used to set a checkpoint

Fault Overhead

$$f(t) = \frac{dF(t)}{dt} \quad \ldots(12)$$

Where,
f (t)= The failure density function
F(t) =The failure distribution function

Failure distribution value

$$F(t) = \int_{-\infty}^{t} f(\tau)d\tau \quad \ldots(13)$$

Where,
f (t)= The failure density function
F(t) =The failure distribution function

Failure expectation value

$$E(t) = \int_{-\infty}^{+\infty} \tau.f(t)d\tau \quad \ldots(14)$$

Where,
E(t)= The expectation function of the number of failures
f (t)= The failure density function

$$T_{fail} = t_{fi} - t_k \quad \ldots(15)$$

Where,
*T*fail =The time interval between the failure point *tfi* and the last checkpoint *tk*
Tk= The time of the *k*th checkpointing
+*tfi* =The time of the *i*th failure occurrence

$$E(T_{fail} \mid t_k < t_{fi} \le t_{k+1}) = \frac{1}{2.p(t)} \quad \ldots(16)$$

Where,
$E(T_{fail} \mid t_k < t_{fi} \le t_{k+1})$=failure expectation distribution value
[t1,tk+1]=check pointing time interval
ρ(t) =The check pointing density function

*T*fail =The time interval between the failure point *tfi* and the last checkpoint *tk*

Fault tolerance overhead

$$T_{rcv} = T_{fail} + T_{chk} \ldots(17)$$

$$T_{rcv} = \frac{1}{2.p(t)} + T_{chk} \ldots(18)$$

Where,
*T*rcv= The time interval between the failure point *tf* and the new checkpoint *tk*+1 after system recovery
*T*fail= The time interval between the failure point *tfi* and the last checkpoint *tk*
*T*chk= The time interval used to set a checkpoint

Total overhead

$$T_{cyc} = T_{free} + T_{tail} + T_{rcv}$$

$$= T_{chk}.(1 + \int_{t1}^{tk} p(\tau)d\tau) + \frac{1}{p(t)} \quad \ldots(19)$$

Where,
*T*cyc= The time interval of a failure cycle
*T*free =The checkpointing overhead during the longest failure-free time interval of the consecutive checkpoints
*T*fail= The time interval between the failure point *tfi* and the last checkpoint *tk*
*T*rcv =The time interval between the failure point *tf* and the new checkpoint *tk*+1 after system recovery

Failure expectation distribution value:

$$E(T_{cyc}) = \int_{0}^{+\infty} (T_{chk}.(1 + \int_{t1}^{tk} p(\tau)d\tau) + \frac{1}{p(t)}).f(t)dt$$

…(20)
Where,
$E(T_{cyc})$= failure expectation distribution value
*T*chk= The time interval used to set a checkpoint
ρ(t) =The checkpointing density function of a continuous checkpoint
f (t)= The failure density function

Cost and Time based Scheduling

A fine-grained pricing model based on the actual allocation of CPU cycles. In other words, a VM with more processing power is associated with a higher monetary cost, and vice versa. In particular, we use a linear pricing model and an exponential pricing model. In the linear pricing model, the cost of using a VM is linearly correlated with the number of CPU cycles. Let ca_{slow} denote the CPU cycle for the slowest VM m_{slow}. If Vc_{base} is the base price charged to m_{slow}, then the cost incurred to execute task v_i on VM m_j can be calculated as follows:

$$c(v_i, m_j) = \sigma \times t(v_i, m_j) \times Vc_{base} \times \left(\frac{ca_{m_j}}{ca_{m_{slow}}}\right)$$

…(21)
Where σ is a random variable used to generate different combinations of VM pricing and capacity. In the exponential pricing model, the cost of VM allocation is calculated as follows:

$$c(v_i, m_j) = \sigma \times t(v_i, m_j) \times Vc_{base} \times \exp^{\frac{ca_{m_j}}{ca_{m_{slow}}}}$$

….(22)
The total monetary cost is computed by

$$C = \sum_{j \in select} c(v_i, m_j) \quad \ldots(23)$$

Incorporating monetary cost into task scheduling adds a layer of complexity. Since we cannot directly compare different scheduling plans with competing objectives, we will

ISSN: 2278 – 7798

use the concept of Pareto dominance to select VMs and carry out comparisons.

Following minimization problem with an objective function for each node $v_i \in V$ as a convex combination of running time, energy and monetary cost:

$$Minimize: \alpha \times T(i,j) + \beta \times C(i,j) for\ all\ m_j \in M \quad ..(24)$$

$$Subject\ to: T(i,j) = \frac{t(v_i, m_j) - t_{min}}{t_{max} - t_{min}}, C(i,j) = \frac{c(v_i, m_j) - c_{min}}{c_{max} - c_{min}}, \alpha + \beta + \gamma = 1 \quad ..(25)$$

where α, β and γ is a cost-efficient factor that represents the user's preference for the execution time and the monetary cost; T (i,j) and C (i,j) represent cost-efficiency ratios of time and costs, respectively; $t_{min(max)}$ and $c_{min(max)}$ are, respectively, the minimum (maximum) execution time and the minimum (maximum) monetary cost of any task scheduling plan.

POSH involves the following three phases:

(a) Weighting Phase: Assign the weights to the nodes and edges in the workflow. The weights assigned to nodes are calculated based on the predicted execution time of the tasks and the weights assigned to edges are calculated based on predicted time of the data transferred between the VMs.

(b) Prioritizing Phase: Create a sorted list of tasks organized in the order how they should be executed. The priority of each task is to be set with the upward priority value, which is equal to the weight of the node plus the execution time of the successors. The task list is generated by sorting the tasks by the descending order of priority.

(c) Mapping Phase: Assign the tasks to the resources based on Pareto dominance. Consecutive tasks are mapped to the resources based on the priority queue. For each task, choose the VM that favors scheduling tasks with low monetary cost to run it. This is done by the pre-defined objective function.

Algorithm 1: Pareto Optimal Scheduling Heuristic

Input: G (N, E): the DAG task graph
M: the set of VMs

1. Compute priority for all nodes $v_i \in V$ by traversing graph upward;
2. Sort v_i in the descending order by its priority value;
3. For each $v_i \in V$ do
4. For each $m_i \in M$ do
5. Compute $\alpha \times T(i,j) + \beta \times C(i,j)$;
6. End
7. End
8. For each task v_i in the ready queue do
9. Assign task v_i to the VM m_j that minimizes the objective function equation 3 of task v_i
10. End

POSH produces a scheduling plan with time slots for task executions and data communications on VMs.

5. RESULTS and DISCUSSIONS:

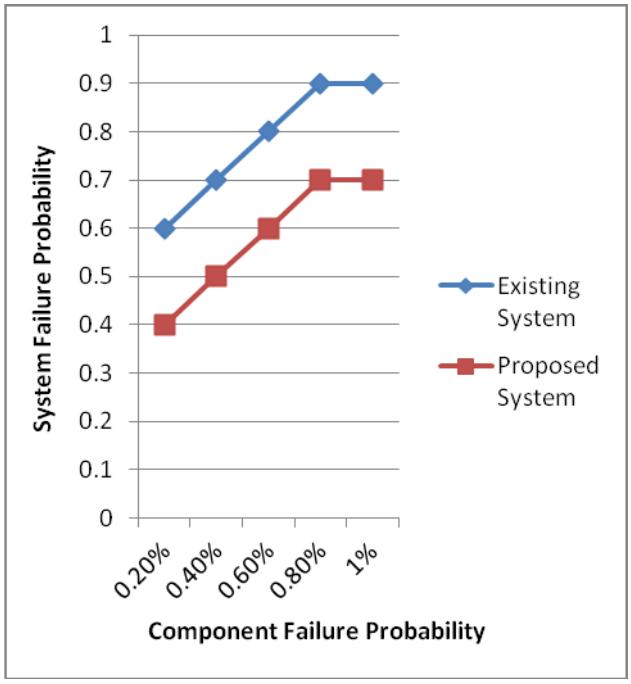

Fig no: 2 Impact of component failure probability

To study the impact of the component failure probability on the system reliability, we compare existing system under failure probability settings of 0.1 to 1 percent with a step value of 0.1 percent. With the increase of component failure probability from 0.1 to 1 percent, calculate the application failure probabilities of two approaches. From the results our proposed system has better than existing approach.

6. CONCLUSION and FUTURE WORK:

virtual machines shall allocate proper host nodes to perform the computation. In this work, we propose performance analysis based resource allocation scheme for the efficient allocation of virtual machines on the cloud infrastructure. , high fault tolerance issue is one of the major obstacles for opening up a new era of high serviceability cloud computing as fault tolerance plays a key role in ensuring cloud serviceability. Fault tolerant service is an essential part of Service Level Objectives (SLOs) in clouds. To achieve high level of cloud serviceability and to meet high level of cloud SLOs, a foolproof fault tolerance strategy is needed. In this work also extends to, the definitions of fault, error, and failure in a cloud are given, and the principles for high fault tolerance objectives are systematically analyzed by referring to the fault tolerance theories suitable for large-scale distributed computing environments. Based on the principles and semantics of cloud fault tolerance, a

dynamic adaptive fault tolerance strategy DAFT is put forward. It includes: (i) analyzing the mathematical relationship between different failure rates and two different fault tolerance strategies, which are check pointing fault tolerance strategy (ii) building a dynamic adaptive check pointing fault tolerance model and a dynamic adaptive replication fault tolerance model by combining the two fault tolerance models together to maximize the serviceability and meet the SLOs; and (iii) evaluating the dynamic adaptive fault tolerance strategy under various conditions in large-scale cloud data centers and consider different system centric parameters, such as fault tolerance degree, fault tolerance overhead, response time, etc. It efficiently and effectively achieves a trade-off for fault tolerance objectives in cloud computing environments. We extend our work to scheduling based on minimizing makespan, cost and energy. Our experimental results shows that our work more efficient for scheduling and resource allocation and improving the resource utilization.

FUTURE WORK:

It can improve the resource utilization to serve a greater number of resource requests at a time without compromising the allocation time.

(i) Examining the efficiency and effectiveness of DAFT fault tolerance strategy in real clouds;

(ii) Evaluating the performance overhead of DAFT fault tolerance strategy underlarge-scale heterogeneous data centers;

(iii) Developing a complete cloud fault tolerance framework using DAFT faulttolerance strategy as a part of cloud services to satisfy the high level of fault tolerance and SLOs requirements;

(iv) Deploying the DAFT fault tolerance strategy on a real cloud computing platform can be made using new optimization techniques (e.g., metaheuristic) and incorporating penalties for violating consumer-provider contracts. To manage monetary costs using cloud services, we would need to manage computing, storage, and network resources with a holistic approach.

REFERENCES:

1. **.** Above the clouds: a Berkeley view of cloud computing by M. Armbrust, et al.

2. . Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility by R. Buyya, C. Shin Yeo, S. Venugopal, J. Broberg, I. Brandic

3. Resource provisioning for cloud computing by J. Wong, G. Iszlai, M.L. Ye Hu,

4. Efficient Resource Provisioning in Compute Clouds via VM Multiplexing by Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet

5**.** Naksinehaboon N, Paun M, Nassar R, Leangsuksun B, Scott S (2009) High performance computing systems with various checkpointing schemes

6.Ling Y, Mi J, Lin X (2001) A variational calculus approach to optimal checkpoint placement

7. Liu H, Jin H, Liao X, Yu C, Xu CZ (2011) Live virtual machine migration via asynchronous replication and state synchronization

8. Khan FG, Qureshi K, Nazir B (2010) Performance evaluation of fault tolerance techniques in grid computing system.

9. D. Warneke, O. Kao, Nephele: efficient parallel data processing in the cloud, in: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, ACM, 2009,

10. H. Topcuoglu, S. Hariri, M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 260–274.

11. J. Yu, R. Buyya, C. Tham, Cost-based scheduling of scientific workflow applications on utility grids, in: First International Conference on e-Science and Grid Computing, 2005, IEEE, 2005, pp. 8–pp.

12. S. Garg, R. Buyya, H. Siegel, Time and cost trade-off management for scheduling parallel applications on utility grids, Future Generation Computer Systems 26 (8) (2010) 1344–1355.

ISSN: 2278 – 7798