Indian Journal of Emerging Electronics in Computer Communications
Vol.3, Issue 2 (2016) Page.577-587
ISSN: 2393-8366

An Efficient Migration of Data in Mobile Access for Multiple Virtual Server through online Vs offline

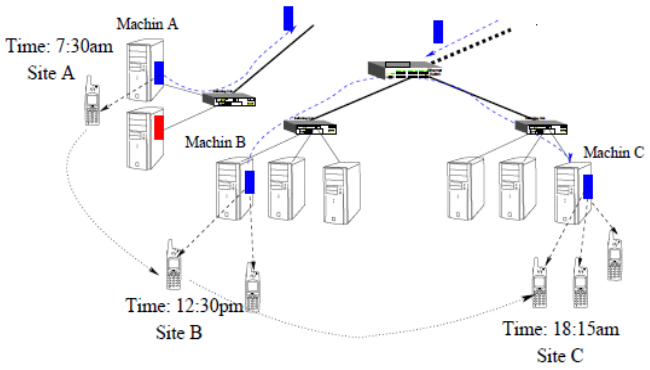
C. Jobin Pinks Anand[1], B. Bhuvaneshwari[2], S. Brindha[3]
[1]*Research Scholar, St.Peter's University, Chennai.*
jobin.anand181@gmail.com
[2]*Asst.Prof. Dept. of Computer Applications, St.Peter's University, Chennai.*
[2]*Asst.Prof. Dept. of Computer Applications, St.Peter's University, Chennai.*

Abstract-- *The nature With the recent advent of cloud computing technologies, a good moving number of our people using applications most of them are from cloud - based services, which shows our project importance and little cost . Two important tasks were done because for migrate the contents to cloud backup, and to correlate the web based services to cloud based web services. Cloud based services for the competing fast world track we don't need time and location for the most importance of both cloud side and mobile applications in the way to do what the user needs with a secures quality of service (QoS). In this work, the cloud-based mobile application through virtual servers represents very good in the statistical reports about its good growth towards top from the year of 2011.*

Keywords: *QOS, cloud based web services, migrate, virtual servers*

I. INTRODUCTION

ADHOC and sensory networks are collapsing into a platform for a large number of application sections in both secured and open platforms. However, the open nature of the wireless connection channels, the lack of lab facility, the fast deployment workouts, and the hostile surfaces where it may be deployed, make them hazardous to a wide range of security threats. Among these threats wormhole attack is very hard to found out because this attack does not inject invalid volumes of traffic into the network. In this paper, a specific type of valid good working security threat knows as the wormhole thread is validated. Migrating the service where now a day's time out to some vantage locations in the network that are close to the clients could be a variable solution if the cost of migration is less than the reaped benefits to both end users and CSPs after the collision achieving such benefits usually requires human work to manually shift the both working server(s) of the requested services over a wide-area network. This is very hard and error oriented, if not worked well.



II. MATERIAL AND METHODS

In this paper, we present a generic optimization framework for dynamic, cost-minimizing migration of content distribution services into a hybrid cloud (i.e., private and public clouds combined), and design a joint content placement and load distribution process that minimizes overall operational cost over time, subject to service response time constraints. co-migration process *Mig algorithm(k)* for multiple servers, each hosting a service replicas .

Mig algorithm(k) is a randomized process with a competitive cost of to collapsed services in a static *n*-nod network where is the maximal ratio of the migration expenses between any pair of neighbor nodes in the network, and where _ and _ represent the large number wired transmission cost and the wireless link cost respectively .We design a joint content placement and load distribution process for dynamic content distribution service deployment in the hybrid cloud. Providers of content distribution services can practically apply it to guide their service migrating and System migrating involves moving a set of instructions or programs, e.g., PLC (programmable logic controller) programs, from one platform to another, minimizing reengineering. Migrating of systems can also involve downtime, while the old system is replaced with a new one. Migration can be from a mainframe computer to more open systems such as Cloud Computing platforms. The motivation for this can be the cost savings. Migrating can be simplified by tools that can automatically convert data from one form to another. The process divides the time into epochs, and compare the cost of our process with that of the optimal process on a per-epoch basis. An epoch consists of one or multiple phases between which *Mig algorithm(k)* migrates at least one server.4 An epoch ends up with the condition that no servers can be migrated. In that case a new epoch starts by resetting related data structures. The process handles each server individually to service each incoming request by gathering the request and accumulating its access cost in related data structures.

2.1 Algorithm

***Mig algorithm(k)*: An Online Co-Migration Process:**

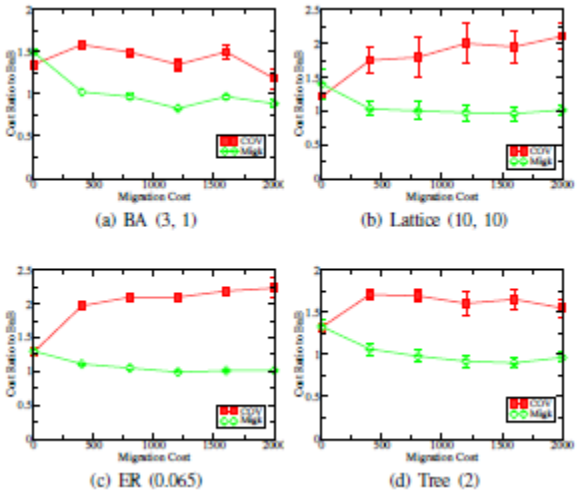
The process divides the time into epochs, and compare the cost of our process with that of the optimal process on a per-epoch basis. An epoch consists of one or multiple phases between which *Mig algorithm(k)* migrates at least one server.4 An epoch ends up with the condition that no servers can be migrated. In that case a new epoch starts by resetting related data structures.

Suppose the κ-server set $S = \{s1, s2, \ldots, s\kappa\}$ is initially located at L, and let $X - x = X \setminus \{x\}$ and $X + x = X \cup \{x\}$, we have the process working as follows;

Currently, many Web services have been deployed by different organizations that are widely distributed over the Internet. These are mostly software services running on fixed hardware resources. When composing multiple services for a system, it is likely that some selected software services are hosted at widely distributed sites. This brings potential performance problems. Sending a service request along with a large quantity of input data across the wide area network can be costly. It increases the network traffic and raises the potential of unexpected delays due to network congestions.

III. PERFORMANCE EVALUATION

Doing those proposed process through expensive simulation-based works. To the last, developed a simulator in Java to create network topologies, generate the access structures and execute the migrating process with connection and without connection, to mobilize the servers to satisfy each generated batch request according to a certain distribution in a time order. Each request is served by its closest server in terms of the shortest path distance. Ties-are-broken arbitrarily.



(a) BA (3, 1) (b) Lattice (10, 10)

(c) ER (0.065) (d) Tree (2)

IV. SERVICE MIGRATION MODEL

Considering an arbitrary *n*-node network *G*(*V*,*E*) as a service infrastructure to provide mobile services. A service has $\kappa \geq 1$ replicas, each running on a virtual machine (VM) (also called *virtual server*). The set of hosting (physical) machines (referred to as servers here after) has a *configuration*, denoted by *L*, which is the specifications of this set of physical machines that are running the VMs. This set of machines are accessed by a sequence of batch requests $\sigma = \sigma 1\sigma 2...\sigma m$ issued from a set of external devices (i.e., mobile terminals). In the online migrating scenario, the requests arrive one at a time. Each of such request is satisfied by triggering the migrating of a set of servers in *L* into an ideal location. On the other hand, in the off-line migrating scenario a migrating is not triggered until a sequence of requests are received, namely, the whole access pattern has been detected and known in advance. It not only benefits the scenarios to proactively schedule the service to facilitate the mobile accesses but also provides a metric to measure the performance of its online counterpart. During a migrating, the service moves through a subset of the virtual servers over time in a live fashion to maximize the efficiency. The required resources for the migrating on the target machines are always assumed to be available. This can be achieve by reservation or pre-configuration of the machines with sufficient resources. Not distinguishing the active and inactive servers like in and all the servers in our model are active. In order to denote the configuration of a subset of servers at *ti* as *Li*. Some frequently used symbols in this and subsequent sections are listed in Table 1 for quick references. Each specific request is routed to the service over a wireless link first to connect the network via a *connect point* and then based on some

process or metrics to reach a service. In order to denote the connection cost (monetary) as μ and the transmission Notation frequently used in model and process. Descriptions , Symbol Meaning n the number of nodes in the network Δmax the large number node degree m the length of a requests sequence Cuv transmission cost bet in order to n node u and v.

V. BRANCH AND BOUND PROCESS PROCESSING

When handling the large installation space and simplifying $wn(Ln)$ in the DP process to find the stable migrating is an overly problematic task. However, the DP process is still valuable as its recurrence matrix contains a lot of viable, and some are relatively good solutions although not stable. Therefore, it is worthwhile to find or not sure for the result in an genuine path. In the sequel, adopt a testing method with local find on the DP matrix whereby creating a multithreaded branch & bound process to overcome the compatible issues and solve this program under a certain condition.

5.1 Method with search tool.

This method is based on DP Repository (4) to do the specification compatible to spontaneously testing the installation space C to obtain the initial κ any specifications, and then upgrading them with local findings according to a revised recurrence of (4). Specifically, suppose the selected κ + 1 configurations are $L00, L10, ..., L\kappa 0$, and the κ servers are initially located at $L00$.

Algorithm 1 Multithreaded Branch&Bound algorithm (BnB)

```
1: procedure BRANCH&BOUND(N_s, N_t, κ, n, σ)
2:     ▷ N_s: # of samples, N_t: # of threads, σ: request seq.
3:     Bound : B ← ∞
4:     Number : N_I = N_s/N_t
5:     while N_s ≥ 0 do
6:         Do PARALLEL
7:             ▷ k ∈ [1...N_t], # of generated samples in SDP
8:                 ▷ there are ⌈N_I/k⌉ rounds for each SDP
9:         N_s ← N_s − k
10:        call SDP(&B, k, κ, n, σ)
11:        END PARALLEL
12:    end while
13:    output(B)                ▷ B stores the final cost value
14: end procedure
```

VI. OPTIMAL SERVER ALGORITHM

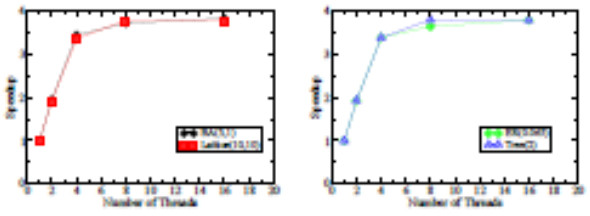
OPT gives the stable side structure property to this program to build the DP repository in this process where its small number expenses solution $wi(Lji)$ at ti can be from the small-expense process to *all* the sub-problems of $L' \in Ci-1$ at $ti-1$ given sever configuration Lji (there could be$(n\kappa)$) possible configurations) for$1 \leq j \leq (n\kappa)$, $wi(Lji)=\min L' \in Ci-1 fwi-1(L')+Expenseacc(L',\sigma i)+Expensemig(L',Lji)g$ where $Ci-1 = \{L1i-1, ..., L(n\_)i-1\}$ is the set of all the possible server configurations at $ti-1$ and σi is the batch request at ti.

Indian Journal of Emerging Electronics in Computer Communications
Vol.3, Issue 2 (2016) Page.577-587
ISSN: 2393-8366

Algorithm 2 Sampling-based DP algorithm (SDP)

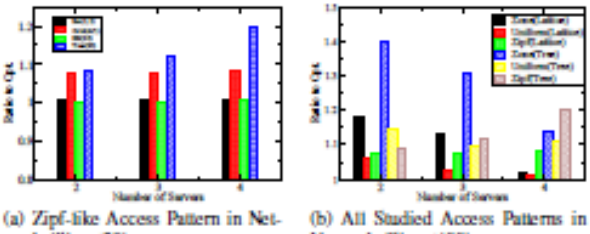
```
1:  procedure SDP(B, k, κ, n, σ)
2:      PriorityQ : Q
3:      Sample : u
4:      ▷ Generate k samples and put into Q, σ = σ_1, ...σ_m
5:      for j ∈ [1, ..., k] do
6:          Sample : s^j ← RandSample(κ, n)
7:          s^j.c ← μ Σ_{t=1}^{|σ|} |σ_t|        ▷ the lower bound of cost
8:          s^j.t ← 1                ▷ σ_t is the next req to be served
9:          Q.enq(s^j)
10:     end for
11:     while Q ≠ ∅ do
12:         u ← Q.deq()
13:         if u.t = |σ| + 1 then
14:                                          ▷ a new upper bound found
15:             if u.c < B then
16:                 B ← u.c                  ▷ update the best bound
17:             end if
18:         else                                        ▷ σ_t is served in u
19:                                          ▷ w_t^u is from (7) for u
20:                                          ▷ u.c is a lower bound of u
21:             u.c ← min_{0≤j≤κ}{w_t^u(L_t^j)} + μ Σ_{t=t+1}^{|σ|} |σ_t|
22:             u.t ← t + 1
23:             if u.c < B then
24:                 Q.enq(u)                        ▷ put back u
25:             end if
26:         end if
27:     end while
28: end procedure
```

In this process, each SDP thread will handle k samples, each having κ randomly selected configurations plus $L0$. All the k samples, each being initialized with its lower bound of service expense to serve σ (Line 7) and the next request in σ to be served (Line 8), are organized in a priority queue with the small number at the top (Line 5-10). Each time, the one at the top is de- queued and used to serve a request in σ (Line 11-12).



(a) BA(3,3) and Lattice(10,10) (b) ER(0.065) and Tree(2)

Fig. 4. Speedup of *Migk* when the number of threads is varied from 1 to 16 for Zipf-like access pattern on all studied networks (Samples=160, configurations=32).

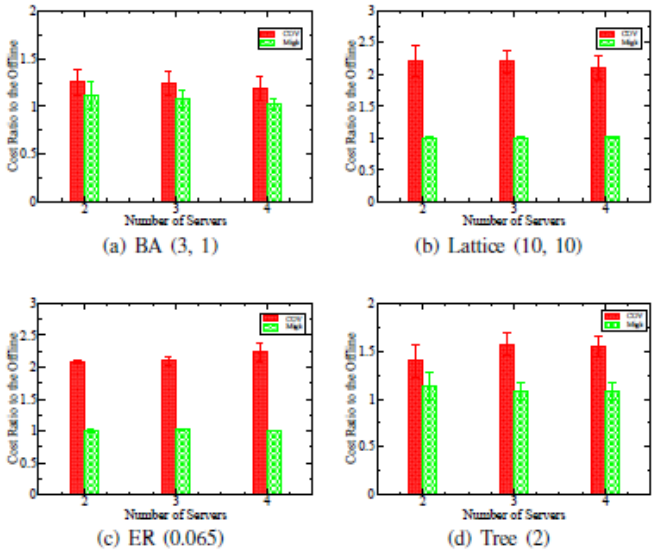


(a) Zipf-like Access Pattern in Network (Size≤20) (b) All Studied Access Patterns in Network (Size≤100)

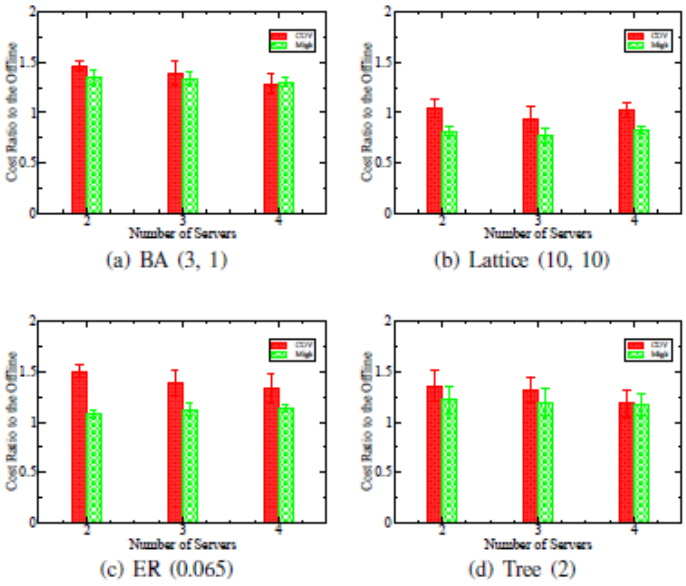
For a particular sample u, the process first checks if $u.t = |\sigma|+1$. If so, the whole sequence σ has been served by u, and a new upper bound is found. If this value is smaller than B, B will be updated with the new upper bound (Line 13-17). Algorithm 2 illustrates the SDP process. As described,there are k samples that are locally generated in each SDP thread. Due to the large sample space, the probability of any two samples (in the same or different threads) that are not the same is as high as $1 - \kappa!/(n\kappa)\kappa$. Therefore, the repeated computation in the process can be ignored.

VII. RESULTS

In this area, in order to concluded that the performance of proposed algorithms in various things. Since the compared online step by step process are evaluated relative to the off-line step by step process, first of all lets measure the performance of the BnB process in forms of its calculation over the conditional DP process where to those problems the future is going to use it as a base to compare the online step by step process. From this, tests every data points in the linear graphs it is calculated over those by mixing the unknown number load into the simulator. To measure the distribution of the values for each data point, in order to also calculate the standard deviation of each data point. However, for clarity of presentation, in order to omitted standard deviation bars on the graphs if most of them are less than 10% of the data point's values.



Performance Studies on the BnB Process Performance performing: The first study of the performance performing of *BnB* with respect to the number of samples. In this thesis, the random number of small tests increases, the performance of the process should also increase accordingly. Test 3 shows that this by giving that how the service expense is reducing that the x axis (the number of tests of all the results) where the number of junks are four. From each example consists of a subset of unknown selected configurations, that the total number of performance of the process is not only matching to the tested size and also relevant to the number of specifications in each test. To calculate this method, all can compare the performance of the process whether the nos of specifications in each time are changed. Through this service expenses of the process decrease as calculations.



Adding number to the processing add-ons, *Mig algorithm(k)* also has its other advantage over that step by step process it is calculating with the *COV*, the processing performance of *Mig algorithm(k)* is actually stable when the migrating expenses are changed by ups and downs. That this is a good work for the service providers that allows the processing expenses were mostly predictable.

VII. CONCLUSION

In this paper, in order to tallied and got new ideas from the migrating problem in online from both online and off-line usages. In online, in order to expressed *Mig algorithm(k)*, an online step by step with a mean competitive ratio of $O(\gamma \log n \min\{1, -+\_\})$ to co-migrate κ servers in a stable n node network with holding $O(\kappa n)$ time, which is good and wise than the previous process in terms of both the competitive ratio and time complexity [16]. In without network , a Branch &bound process, *BnB*, which is based on Data process techniques and processing on our testing methods to genuinely calculate the better results when the total no. of servers is upper-installed by $O(\log n \log(1+\Delta))$, a practical case in reality. Our pre defined results given that of *BnB* on average are stable within a wise of the better result across all our test processing cases. Group proposed Mig algorithm(k), an connection with randomized process with a mean good ratio of $O(\gamma \log n \min\{1\kappa, \mu\lambda+\mu\})$to co-migrate κ servers n-node within O (κn)time. In which is better than the last results in terms of both the ratio and time duration.

REFERENCES

[1]"Good technology's 2nd annual state of BYOD report," 2013, http://media.www1.good.com/documents/Good-BYOD-Report-2013.pdf.
[2]S.-C. Lee, E. Lee, W. Choi, and U.-M. Kim, "Extracting temporal behavior patterns of mobile user," in Networked Computing and Advanced Information Management, 2008. NCM '08. Fourth International Conference on, vol. 2, 2008, pp. 455–462.

[3]M. Bienkowski, A. Feldmann, J. Grassler, G. Schaffrath, and S. Schmid, "The wide-area virtual service migration problem: A competitive analysis approach," IEEE/ACM Trans. Netw., vol. 22, no. 1, pp. 165–178, Feb. 2014.
[4]"Netflix," 2014, http://www.netflix.com/.
[5]"Lions gate," 2014, http://www.lionsgate.com/.
[6]"Virtualappliance,",2014http://searchservervirtualization.techtarget. com/definition/virtual-appliance.
[7]C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2, ser. NSDI'05, 2005, pp. 273–286.
[8]M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," SIGOPS Oper. Syst. Rev., vol. 43, no. 3, pp. 14–26, Jul. 2009.
[9]R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schi¨oberg, "Live wide area migration of virtual machines including local persistent state," in Proceedings of the 3rd international conference on Virtual execution environments, ser. VEE '07, 2007, pp. 169–179.
[10]H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu, "Live migration of virtual machine based on full system trace and replay," in Proceedings of the 18th ACM international
symposium on High performance distributed computing, ser. HPDC '09, 2009, pp. 101–110.
[11]F. Lai, Y.-S. Wu, and Y.-J. Cheng, "Exploiting neighborhood similarity for virtual machine migration over wide-area network," in Software Security and Reliability (SERE), 2013 IEEE 7th International Conference on, 2013, pp. 149–158.
[12]S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "Vmflock: virtual machine co-migration for the cloud," in Proceedings of the 20th international symposium on High performance distributed computing, 2011, pp. 159–170.
[13]T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe, "Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines," in Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, ser. VEE '11, 2011, pp. 121–132.
[14]P. Riteau, C. Morin, and T. Priol, "Shrinker: genuine live migration of virtual clusters over wide area networks," Concurrency and Computation: Practice and Experience, vol. 25, no. 4, pp. 541–555, 2013.
[15]J. Schneider and S. Schmid, "Optimal bounds for online page migration with generalized migration costs," in INFOCOM, 2013 Proceedings