# Redundant Virtual Machines Management in Virtualized Cloud Platform[1]

Fan Zhang[1], Junwei Cao[2,3], Hong Cai[4], Lianchen Liu[1,3] , Cheng Wu[1,3]

*[1]National CIMS Engineering and Research Center, Department of Automation*

*Tsinghua University, Beijing 100084, P. R. China*

*[2]Research Institute of Information Technology, Tsinghua University, Beijing 100084, P. R. China*

*[3]Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, P. R. China*

*[4] IBM China Software Development Lab*

*Corresponding Author:* jcao@tsinghua.edu.cn

***Abstract***

**Selecting and utilizing proper virtual machines in a virtualized cloud platform to achieve high availability, throughput, reliability, as well as low cost and makespan is very important. The importance lies in the adaptive resource provisioning to satisfy variant of workloads. An Adaptive Accessing Aware Algorithm (A5) is proposed in this paper to deal with this conflicting objective optimization problem. The main strategy of A5 is selecting adaptive upper/lower bound of service capacity to decide the time for scheduling redundant virtual machines and a Pareto-front based multi-objective optimization method to decide the number of scheduling virtual machines. We carried out experiments in simulation, which show that A5 can achieve much higher performance improvements in four different workload testing environments, compared with other three commonly used methods.**

***Keywords— Pareto-front; redundancy; upper/lower bound; virtualized cloud platform; virtual machine***

[1] Manuscript submitted to International Journal on Modeling, Simulation and Scientific Computing on 24 July 2010; revised version submitted on 25 Sept. 2010. Part of this work is performed when Fan Zhang is a intern student in IBM China Development Lab supported by 2010-2011 IBM Ph.D. Fellowship.

## I. Introduction

### A. *Research Background & Motivation*

The advent of Cloud Computing [1], which leverages virtual clusters to extremely large scale, widely distributed, commodity-computer data centers and consistent service provisions are requiring reliability, availability, throughput and lower cost. However, the optimization over all these conflicting metrics is difficult. For example, improving throughput leads to the scaling the cost of using more number of virtual machines.

Recently we have witnessed lots of proposals to deal with this problem. Failover clustering [2] combines one or more nodes, or servers, with two or more shared disks. If one server fails to provide service, failover strategy migrate the running applications to other server in the disk groups. Data Redundancy [3] is also very popular to provide high availability solutions. Hadoop [4], an open source project and a light-weight implementation of Google File System (GFS) [5], replicate each file twice to ensure availability.

The major challenges are concluded as follows.

(1)**Low utilization**. There is an inborn conflicting between redundancy and utilization. Redundant virtual machines (VMs) are used as replications to provide sustainable services for high availability. Low utilization problem emerges when workloads are relatively light since lots of VMs are idle. How to provide a flexible strategy to effectively utilize these resources adaptively is one of the technical challenges.

(2)**High cost**. Replications of data require software and hardware investment and maintenance cost. How to minimize the overall cost and keep the system availability at an acceptable level to adapt to variant system workloads is another challenge.

(3)**Low reliability**. The reliability of the system may not grow linearly with the redundancy of the system which may result in extra cost. using How to ensure high reliability adaptively with consideration of cost is also difficult.

An Adaptive Accessing Aware Algorithm (A5), is proposed in this work, aiming at addressing the above issues. The major contributions include:

(1)**Global optimization of throughput, availability, reliability and cost aware.** Reliability and cost of different redundant VMs have different impacts on throughput and availability. A selection strategy based on achieving Pareto-front with multi-objective programming is brought in to address this global performance metrics balance problem.

(2)**Adaptive accessing aware**. Arrival/departure rate of workloads influence service capacity differently. Different requests exert different pressure on system performance since the various resources they require to use and their own complexity are also different. A5 is proposed to adaptively adjust itself for this situation and automatically change its scheduling policy to optimize the objectives in (1).

(3)**Adaptive upper and lower bounds of redundant VMs.** When the Cloud platform is overburden (or over idle), an adaptive threshold of choosing available redundant VMs to enrich the active redundant list (or in an opposite direction) is effectively calculated to take full advantage of redundant resources. We use a heuristic method to balance low service capacity with the cost of frequently schedule in redundant VMs for lower bound selection. Also, we balance high service cost with the cost of frequently schedule out redundant VMs for upper bound selection.

## *B. Related Work*

Redundant systems [6] have long been used as an effective strategy to deal with low reliability, error prone problems not only in computational cluster systems, but also in mechanical engineering, electrical system design, etc. The redundant resource allocation is an NP-hard problem [7, 8, 9] which makes it not able to be solved by polynomial algorithm. H. Lee [10], et al made a comprehensive comparison of two famous strategies: max-min approach and the Nakagawa and Nakashima method, to conclude the advantage and disadvantage of them. Performance of Nakagawa and Nakashima method depends strongly on initial redundant strategy, which limits its implementation in real cluster applications. Based on the max-min method, our A5 is an adaptive extension of this method, which includes objectives more than only reliability. A. Azaron, et al. [11] proposed a new methodology for the reliability optimization of a k dissimilar-unit (or heterogeneous) non-repairable cold-standby redundant VMs and apply the shortest path technique in stochastic networks to evaluate the reliability. The evaluation of reliability technique is similar in A5. Different from this work, A5 tends to find a proper interval to ensure the selected active redundant system running with a highest reliability. D. Salazar, et al [12] detailed the redundancy strategies in constrained multi-objective reliability problems, and then uses evolutionary algorithms (MOEA) to identify a set of optimal solutions (Pareto-front) to ensure the selected solutions not to be dominated by any other possible solutions. A5 absorbs the advantage of Pareto-front which order the redundant VMs based on their reliability, cost and service capacity. Most of these works are more theoretically based, and many mature methodologies can be rectified to suit for redundant cluster systems in our A5 as described above.

Other than that, many research on computational resource allocation and cluster reliability awareness provide lots of solutions to deal with similar problems. Redundant strategy has been proven to be very efficient in P2P file sharing system [13] named Hyper-Chord. Different from that, our proposed A5 is not a structured architecture. D. W. Coit [14] researched on many cold-standby redundancies to maximize system reliability from theory and proposed its implementation in cost-efficient engineering designs. B. Hong, et al [15, 16, 17] also consider adaptive resource allocation problem with equal-sized task to maximize the throughput. Instead, A5 algorithm tends to consider tasks with unequal sizes (which brings in different pressure on system performance), and uses more performance metrics to heuristically and adaptively dispatch the redundant VMs.

Grid workflow scheduling problems for performance improvement are investigated in [18, 19, 20, 21]. Our A5 differentiate from these works in that it mainly deals with virtualized cloud platform, in which virtual resources are provisioned on demand. Most of the previous related works for grid or P2P resource scheduling are based on the fully utilization of all the resources. This is a very different point.

The remainder of the paper is organized as follows. Section II introduces the redundant virtualized Cloud platform model and traditionally used max-min optimization strategy. Section III introduces the rationale of A5. Section IV gives out A5 algorithm and the analysis. Abundant experiments and performance evaluation are conducted in section V to show the efficiency of A5 over other methods. At last, some conclusions are made and future works for improvements are listed in section VI.

## II. REDUNDANT VIRTUALIZED CLOUD PLATFORM

### A. *Preliminaries of Dynamic Redundant VMs*

In Fig. 1, we depict the Virtualized Cloud platform built with 4 virtual clusters over 3 physical clusters. Each physical cluster consists of a number of interconnected servers, represented by the rectangular boxes with 3 different shadings for the 3 physical clusters shown. The virtual machines are implemented on the servers (physical machines). Each virtual cluster can be formed with either physical machines or VMs hosted by multiple physical clusters. The boundaries of the virtual clusters are shown with 4 dot/dash-line boxes. The provisioning of VMs to a virtual cluster can be dynamically done upon user demands.

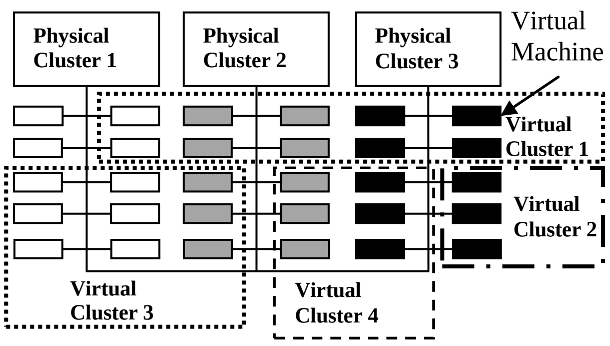


**Figure 1. A virtualized Cloud platform with 4 virtual clusters deployed on 3 physical clusters**

Theoretical notations we use in our experiment are firstly summarized in Table 1. Fig. 2 shows the workload generation scenario and sketch of our scheduling systems.

TABLE 1. NOTATIONS OF USE

| Notations | Descriptions |
|---|---|
| $s$ | Number of physical clusters |
| $n_i$ | Number of total VMs in physical cluster $i$ ($i \in [1,s]$) |
| $m_i$ | Number of available VMs for physical cluster $i$ |
| $x_{ij}$ | Number of VMs in physical cluster $i$ and we choose VM $j$ to use |
| $C$ | Maximum allowable system cost |
| $c_{ij}$, $r_{ij}$ | Cost, weight and reliability of choosing virtual machine $j$ in physical cluster $i$ |
| $R_s$, $T_s$, $C_s$ | Overall reliability, throughput and cost |
| $R_i(x_i)$,$T_i(x_i)$ $C_i(x_i)$ | Reliability, throughput and cost of virtual machine $x_i$ |

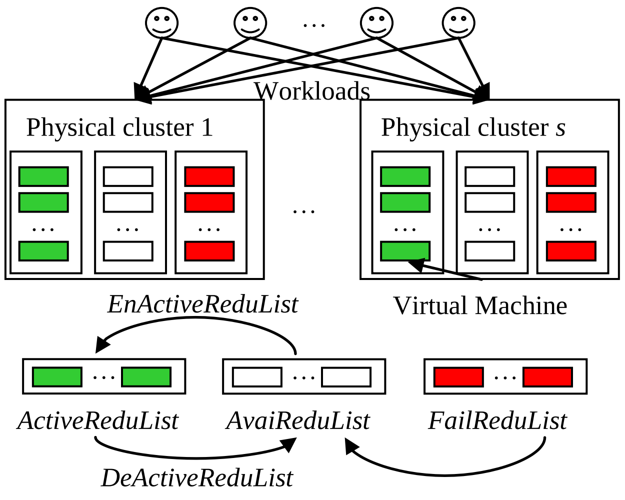


**Figure 2. Workload generation scenario and sketch of our scheduling systems**

There are three lists in our experiments.

1. Active redundant list (*ActiveReduList*): It holds redundant VMs that are currently in use represented by the green rectangular box in Fig. 2.

2. Available redundant list (*AvaiReduList*): It holds redundant VMs that are prepared to use represented by the white rectangular box in Fig. 2. We schedule redundant VMs into *ActiveReduList* from *AvaiReduList* when the cloud platform is overburden (*EnActiveReduList*) or release extra redundant VMs when it is over idle (*DeActiveReduList*).

3. Failed redundant list (*FailReduList*): It holds VMs that are failed currently or permanently represented by the red rectangular box in Fig. 2.

Based on these analysis, we can see that the workloads generated by all terminal users feeding into physical cluster $i$ ($i \in [1,s]$) and its redundant virtual servers $ij$ ($j \in [1, m_i]$) form a queue service model. But this queue characterizes itself in the following two aspects, which is different from traditional queue theory.

(1) Rather than one workload have to be pended until its previous one finished its service, Multi-thread processing allows all workloads arrived at one physical cluster could be executed concurrently, which violates the most basic premise in queue theory[23].

(2) Availability of VM violates another basic premise that service provider should be always reliable. It is very common that some services are possibly to be terminated because of the potential failures of VMs.

Different from queue theory that service provider has stable service capacity (*SC*), our system provides dynamic service capacity $SC(t)$ in that there are some reliability concerns and variable service providers' scenarios. Too high *SC* means too many redundant VMs, which brings in cost problem. On the other hand, too low *SC* brings in problems of low throughput.

## B. *Max-Min Approach*

The most commonly used max-min approach are first developed by Ramirez-Marquez et al [22], which takes into account the reliability of the whole system as the optimizing objective. It is constrained by the cost since its basic premise is that a subsystem with redundant VMs has a higher, or at least an equal reliability than the original system. Based on these results, the max-min approach for redundant system can be formulated as follows (we use reliability as an example):

$$\max_x(R_S)=\max_x\prod_i(R_i(x_i)) \qquad (1)$$

Subject to

$$\sum_i\sum_j c_{ij}x_{ij}\le C$$

$$x_{ij}\in Z^+$$

The problem of maximizing the reliability of each component subject to cost and weight constraint equals to maximizing the reliability of minimal reliable component. In this way, the objective function can be reformulated as

$$\max_x(R_S)=\max_x\left\{\min_i\left\{1-\prod_j(1-r_{ij})^{x_{ij}}\right\}\right\} \qquad (2)$$

This is an integer programming problem and there are many existing methods, such as interior point cutting plane method or branch-and-bound algorithm, to solve it. Also we can transform the constrained optimization problem into unconstrained problem by bring in a penalty function.

## III. ADAPTIVE ACCESSING AWARE STRATEGY

### A. *Preliminaries of A5*

A5 addresses a multi-objective optimization problem, which includes four performance metrics: throughput, cost, makespan and reliability. Detailed introduction of these metrics are given in section III.B.

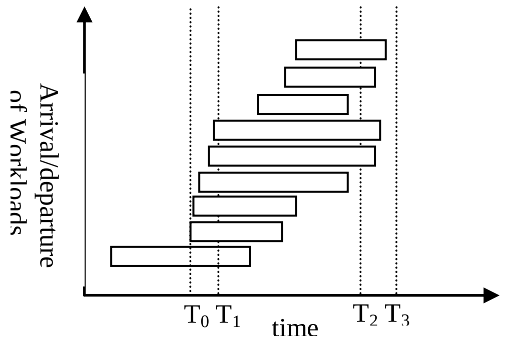


**Figure 4. Arrival and departure of workloads, the length of rectangular box represent its *EET*.**

The main principle of A5 is demonstrated in Fig. 4. During [$T_0$, $T_1$], a dense arrival rate with long *Expected Execution Time* (*EET*) requests, which leads to A5 to choose proper redundant VMs, *EnActiveReduList*, to alleviate the burden. During [$T_2$, $T_3$], a dense departure rate leads A5 to release redundant VMs, *DeActiveReduList*, to ensure a low cost. Problem arises that when and how should we do this. In the following sections, an upper/lower bound selection method answers the "when" problem and a Pareto-front based multi-objective optimization scheduling answers the "how" problem.

In view of the above analysis, the following concepts and analysis are under consideration when we design redundant strategy.

(1) **Real time service capacity (SC(CurTime))**. Based on X. Yang's work [24] on service capacity in P2P systems, the increase of peer nodes' join can increase system performance from $\beta$ to $\beta m$ as the replica increase $m$ times. This can be applied here straightforward since we share a similar scenario. If the current *SC* of our platform at time $t$ is $SC(t)$, another redundant VM with service capacity $SC'(t)$ is chosen to enlist in need, then the service capacity can be simplified as ($SC(t)$+ $SC'(t)$). Similar rule can be applied in delist.

(2) **Adaptive upper and lower bound of service capacity.** Upper/lower bound at current time, which are denoted as *UpperBound(CurTime)* and *LowerBound(CurTime)* respectively, are used for deciding when to increase/decrease redundant VMs. Suppose the system is preferably to run at an

average SC as shown in Fig. 5, if *SC(CurTime)* > *UpperBound(CurTime)*, it means that extra service capacity is used and we decide to do *DeActiveReduList*. On the other hand, if *SC(CurTime)* < *LowerBound(CurTime)*, more VMs are needed because of low service capacity. The major difficulty of making the decision is how to adaptively decide the two bounds. If *UpperBound(CurTime)* is too high as shown in 1, it will potentially enlist many redundant VMs until it is the time to release, which makes the system runs at a very high *SC* but leads to unnecessary running cost. On the contrary, if it is set too low as shown in 2, the bound is easy to go over, which leads to frequently DeActiveReduList() and brings other cost overhead. Similar dilemma exist in the selection of *LowerBound(CurTime)*. Proper and adaptive decision of the two bounds can decrease total cost and increase throughput significantly.

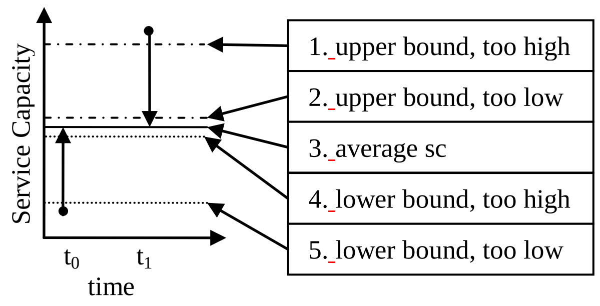


**Figure 5. Demonstration of upper/lower-bound**

(3) **En/De-ActiveReduList strategy**. Three metrics, namely *SC*, *cost* and *reliability*, are considered altogether. In Fig. 5, we want to choose a combination of redundant VMs (*EnActiveReduList*) that can lead to the *SC* increasing to a value very close to *average SC* with low cost and high reliability. The optimization problem can be formulated like this:

$$\left\{\min\left(\left|asc-\left(SC(t_0)+\sum_{i=1}^{s}SC(s)\right)\right|\right),\min\left(\sum_{i=1}^{s}Cost(s)\right),\max\left(\min\left(Reli(s)\right)\right)\right\} \quad (3)$$

*s* is the index of the selected redundant VMs in the available redundant list and *asc* denotes the average service capacity. It is a multi-objective optimization problem. *DeActiveReduList* can be similarly formulated as follows (suppose the current time is $t_1$),

$$\left\{\min\left(\left|asc-\left(SC(t_1)-\sum_{i=1}^{s}SC(s)\right)\right|\right),\max\left(\sum_{i=1}^{s}Cost(s)\right),\min\left(\max\left(\mathrm{Re}\,li(s)\right)\right)\right\} \quad (4)$$

*s* is the index of the selected redundant VMs in active redundant list.

(4) **Pareto-front based scheduling**. Mathematically, achieving Pareto-front is defined as a multi-objective optimization problem. Supposing our bi-objective optimization problem is **min ($J_1$(θ), $J_2$(θ))** and we plot all **($J_1$, $J_2$)** pairs in Fig. 6. Intuitively we can see red dots $\{\ell_1\}$ are better, at least not worse, than other white dots from both $\mathbf{J_1}$ and $\mathbf{J_2}$. Thus they are called Pareto-front. If we remove $\{\ell_1\}$, $\{\ell_2\}$ forms the Pareto-front of the remaining space. In this way, we can get a series of ordered layers $\{\ell_1\},\{\ell_2\},\ldots$. This rule can also be similarly applied in more objectives.

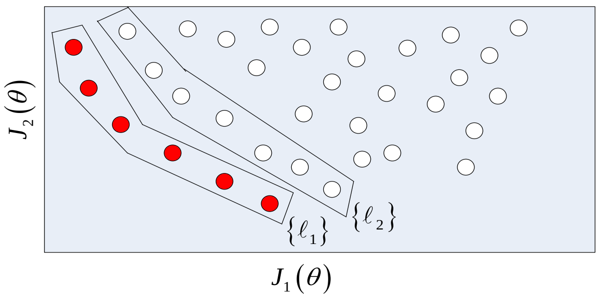


**Figure 6. Demonstration of Pareto-optimal sets**

## B. *Optimization Metrics Analysis*

Based on the above analysis, we reformulate the optimization objectives as follows:

$$\{\max(R(ET)),\min(M(ET)),\max(T(ET)),\min(C(ET))\}$$

*ET*, *R*, *M*, *T*, *C* are short for *Experimental Time span*, *Reliability*, *Makespan*, *Throughput* and *Cost* respectively. The definition of each metric is as follows:

**a.** $R(ET)$: *average*(*max*(*Reli*(*ActiveReduList*(*i*)) *at time t*)), $i \in [1$, *length* (*ActiveReduList*) *at time t*].

**b.** $M(ET)$ is defined as: $\forall\ t = [t_0, t_1]$, if *ActiveReduList* during t is not null, $M(T) = M(T) + t$.

**c.** $T(ET)$ is defined as: *Number of Workloads/M*(*T*).

**d.** $C(ET)$ is defined as: $\forall\ t = [t_0, t_1]$, if *ActiveReduList* at $t$ is not null, $\forall\ n$ in *ActiveReduList*, $C(ET) = C(ET) + Cost(ActiveReduList(n)) * [t_1 - t_0]$. $\forall\ i$ = *En/De-ActiveReduList*() *at time t'*, $C(ET) = C(ET) + SchedCost(i)$.

**Explanation of the four performance metrics**:

If *ActiveReduList* is not empty at time *t*, the reliability of the whole system is decided by the maximum reliability of the VMs in the list. A5 tries to find an average best during the *ET* as shown in **a**.

Makespan is defined as the aggregated timespan that at least one vm is being in use. If *ActiveReduList* is not empty at time *t* then it means there are still workloads under processing. The makespan should include all this part of time as shown in **b**.

Throughput is defined by the total workloads divided by the total makespan, which shows how many workloads are processed on average during the whole experimental time as shown in **c**.

In **d**, we can see that the cost is composed of two parts. The first part is the *service cost*, which equals to how long each redundant VM *n* is used ($[t_1 - t_0]$) and its unit cost (*Cost*(*ActiveReduList*(*n*))). The second part is the *scheduling cost*, which includes scheduling in/out cost of redundant system *i* to/from ActiveReduList (*SchedCost*(*i*)).

## IV. Adaptive Accessing Aware Algorithm

Algorithm 1 below is the main body of A5. It uses reliability, cost, throughput and makespan based on definition in section III. At the beginning of the experiment, the workload increases with large bulks of requests, which decrease service capacity under the *LowerBound*(*CurTime*), A5 then enriches the active redundant list by invoking *EnActiveReduList*(). As the experiment goes on, more requests are finished. This alleviate service capacity above *UpperBound*(*CurTime*), A5 invokes *DeActiveReduList*() to avoid too much cost problem. How to find a proper pair of *UpperBound/LowerBound* dynamically is demonstrated in Algorithm 2, then the strategies of enriching/releasing redundant VMs are shown in Algorithm 3.

## A. *A5 Algorithm*

Algorithm 1. A5 Algorithm

```
1.  Initialize Flag, WLoadNum, Start(i), Leave(i), i∈[1, WLoadNum];
2.  Initialize ReduNum, ActiveReduList = {}; AvaiReduList = {11, 12,…,nm_n}; FailReduList = {};
    Capa(AvaiReduList[k]); Reli(AvaiReduList(k)); Cost(AvaiReduList(k)); k∈[1, ReduNum];
3.  Initialize CurSC, CurLoc, CurCost, CurReli, Pretime, PreSC, CurM;
4.  while(CurLoc <= Leave(WLoadNum))            // experiment not end
5.    PreSC = CurSC;        //get PreSc to calculate SCInc in Algorithm 2
6.    if(ActiveReduList != null)
7.      for each (j in [1, length(ActiveReduList)])
8.        if(!Avai[ActiveReduList[j]])
9.        EnList(FailReduList, ActiveReduList[j]);
10.            CurSC –= Capa[ActiveReduList[j]];
11.            end if                    //ActiveReduList[j] failed, decrease its sc
12.          end for
13.        end if                        //record fail node at CurTime and CurSC
14.        for each (i in [1, WLoadNum])
             //new workload i arrive, decreasing current service capacity
15.          if(Start(i) == CurTime) CurSC –= EET[i]; end if
             //old workload i depart, increasing current service capacity
16.          if(Leave(i) == CurTime) CurSC += EET[i]; end if
           //makespan calculation flag according to b. in section III. B.
17.          if (CurTime > Start(i) && CurTime < Leave(i)) Flag = 1; end if
18.        end for  //find next start or leave request and their influence on CurSC
19.        while(CurSC < LowerBound(CurTime)) // invoke algorithm 2
20.          EnActiveReduList();          //Overload, invoke algorithm 3
21.        end while
22.        while(CurSC > UpperBound(CurTime)) // invoke algorithm 2
23.          DeActiveReduList();        //over idle, invoke algorithm 3
24.        end while
25.        if (Flag == 1) CurM += 1 / CurSC; end if    //makespan calculation
//CurLoc goes for one step unit and calculate the time consuming
26.        CurLoc++; PreTime = CurTime;  CurTime += 1 / CurSC;
```

```
27.         for each (j in [1, length(ActiveReduList)])
28.           TotalCost += Cost(ActiveReduList [j]) / CurSC
29.         end for   //Find TotalCost and Total Reliability currently
30.         TotalReli += max(Reli(ActiveReduList))
    //according to a. in section III.B.
31.       end while
32.       MakeSpan = CurM;      //summarize to get makespan and throughput
33.       Throughput=WLoadNum/MakeSpan;//according to c.in section III.B
```

Algorithm 2. Upper/LowerBound(CurTime)

```
34. TimeInc = CurTime – Pretime;
35. SCInc = CurSC – PreSC;
    //If SCInc < 0, SC decreased, Upper/Lower bound decrease accordingly, else Upper/Lower
    bound increase accordingly
36.      UpperBound = UpperBound + SCInc/ (TimeInc*f);
37.      LowerBound = LowerBound + SCInc/ (TimeInc*f);
38.     Asc = (UpperBound + LowerBound)/2;
39. end
```

Algorithm 3. En/De-ActiveReduList()

```
40.       {{1},{2},…,{j}/{k}}=ParetoFront(Avai/ActiveReduList)
41. for each ({n} in {{1},…,{j}/{k}})
42.     if ({n} != null)//{n} contains the nodes in n^th layer of pareto front
43.       order Avai/Active-ReduList by Capa{n} ascending
44.       else if(∀ n_1, n_2∈{n} && Capa[n_1]== Capa[n_2])
45.              order Avai/Active-ReduList by Cost{n} descending.
46.         end if
47.       else
48.           order Avai/Active-ReduList by Reli{n} ascending.
49.           s = SelectFirstof({n});                //the best one in list
50.           Delist(Avai/Active-ReduList, s);
51.           Enlist(Active/Avai-ReduList, s);
52.           TotalCost += (SchedCost(AvaiReduList(s)) + SchedCost(ActiveReduList(s)));
    //cost of schedule in/out
53.           CurSC = CurSC +/– Capa[s];         //change sc
54.     end if
55. end for
```

{1},{2},…,{j}/{k} are ordered layers of *Avai/Active-ReduList* respectively based on (5) in section III.A. For the redundant VMs of one Pareto-front layer, the selection are ordered by service capacity, cost and reliability. In this way, we can guarantee an optimum selection of preferable suitable VMs.

A flowchart in Fig. 7 is used to illustrate the application procedure of our A5 algorithm.

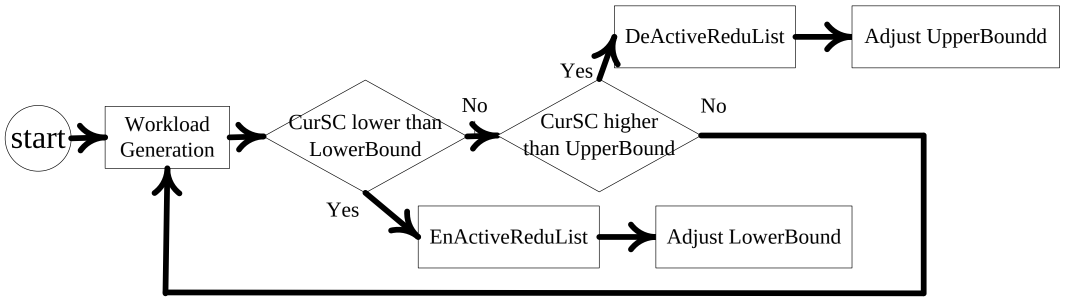


**Figure 7. A flowchart to illustrate the algorithm**

## B. *Algorithm Analysis*

In our algorithm, the current time is *CurTime* and the service capacity is *CurSC*. Each *CurLoc* goes one step forward and the time it takes is different because of different *CurSC*. There are three points that affect *CurSC*.

(1)**Fail of running VM *i***. If VM *i* fails to provide service, the system would decrease service capacity by *Capa*[*i*], which is shown in line 6 to 13. To make things simpler, we unify the unit of *Capa*[*i*] into *CurSC* to make them directly addable/subtractable as shown in line 10 of Algorithm 1 and line 53 of Algorithm 3.

(2)**Arrive/Leave of requests**. Newly arrived requests decrease service capacity depending on their *EET*. The algorithm simulates this scenario by automatically generating workloads with different *EET* to show their different impacts on service capacity. Similar with (1), we also unify the unit of requests' impact *EET* into service capacity to make then addable/subtractable as shown in line 15, 16 of Algorithm 1.

(3)**The increase/decrease of redundant VM list**. As shown in algorithm 3, if we schedule redundant

VM *s* into *ActiveReduList*, the *CurSC* would increase by *Capa*[*s*] and vice versa. We bring in the concept of Pareto-front to order the performance metrics, namely *sc*, *cost* and *reliability* as shown in Algorithm 3, into different layers (line 40, algorithm 3). We then select from the first layer on, which contains VMs with better sc (line 43), cost (line 45) or reliability (line 48). This is in the sense of pareto optimal as the red dots compared with the white dots in Fig. 6.

The decision of when to increase/decrease redundant VMs depends on LowerBound/UpperBound. The two bounds are decided by the rate of fluctuation of service capacity. As shown in Algorithm 2. Here we only introduce the situation that should increase redundant VMs. Intuitively, *LowerBound* should decrease, at least not increase, as *CurSC* decreased, but it should not be faster than, or even equal to the rate of *CurSC* since if so, redundant VMs would never be used. We heuristically divide the rate by f ($f > 1$) in line 36 and 37, to deal with this problem. After the setting of *LowerBound*, *UpperBound* changes with the same ratio. Abundant experiments show that the selection of f differs with the four experimental environments.

## V. PERFORMANCE EVALUATION

Simulation test bed is set with 100 redundant VMs that can provide service simultaneously. The service capacity is strongly dependent on CPU rate and network condition which is similar in work [25]. We then generated cost and reliability of each redundant system that is proportional to its service capacity, which is reasonable in real scenarios. To make this simulation more practical, the arrival frequency of each workload is a long range that shows their different impacts on the performance metrics. Besides that, *EET* is properly randomly generated based on the service capacity and reliability of each redundant VM as well as their arrival frequencies.

The workloads can be categorized into four groups. In the first group, the workload has low frequency and low *EET* (*LFLE*), which means each request comes very occasionally and each one consumes a minor fraction of resource. The second group has workload with high frequency and low *EET* (*HFLE*). The third group has workload with low frequency and high *EET* (*LFHE*), which means the interval arrival time of two workloads are long but each request is resource consuming. The last group has workload with high frequency and high *EET* (*HFHE*), which give the server most pressure.

The experiments are carried out using two computers separately. One is Windows XP with Intel Core 2 Duo T5870, 2G RAM and the other is Windows XP with Intel Pentium D 2.8GHz, 2G RAM. The final results are averaged by 10 experimental results.

## A. *Performance of adaptive Upper/Lower bound*

Adaptive upper/Lower bound selection is an effective way to automatically adjust redundant strategy to deal with the frequently changing workloads. The workloads are generated in uniform distribution in our experiment and are set in Table 2:

TABLE 2. WORKLOAD SETTINGS

| | Inter-arrival time | EET |
|---|---|---|
| LFLE | U(25,30) | U(3,5) |
| LFHE | U(25,30) | U(35,50) |
| HFLE | U(3,5) | U(3,5) |
| HFHE | U(3,5) | U(35,50) |

Here we discuss a scenario that cost and reliability are proportionally correlated with service capacity. Service capacities of all redundant VMs are normalized to be at a mean value of 100 with standard deviation of 20, which is concluded by our experiments with testing a same application on different servers. The cost is normalized with the same mean of service capacity, but with standard deviation of 50. The reliabilities of them are set with a mean of 90% and a standard deviation of 10%. We performed many fixed upper/lower bound (generally upper bound is set with a mean of 1.8, lower bound is set with a mean of 1.6, and *average SC* is set with a mean of 1.7) experiments and averaged the results. The initial setting of the adaptive strategy is the same with the mean of the fixed setting. The results are shown in Fig. 8.

We can see from the Fig. 8 that with a fixed (non-adaptive) upper/lower bound choice, the performance in all four experiments are worse than adaptive selection. The improvements are even noticeable in cost because of line 45 in Algorithm 3 since cost is the second most important criterion for redundant strategy. The most important one is the deviation from the *average SC* (the first item of (3) and (4)), which is decisive and a guarantee that A5 can make sure the *average SC* not too high or too low, and it leads to better performance in the four metrics under the four experimental environments.

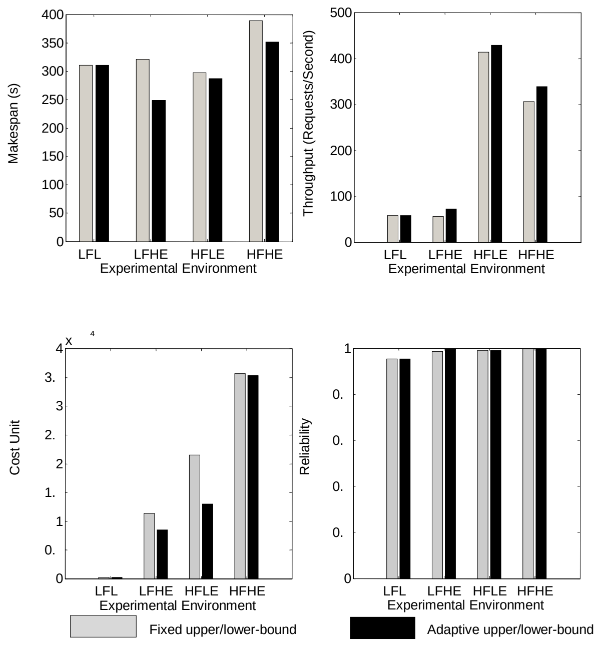
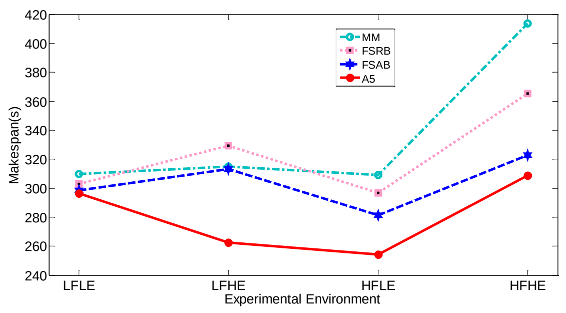


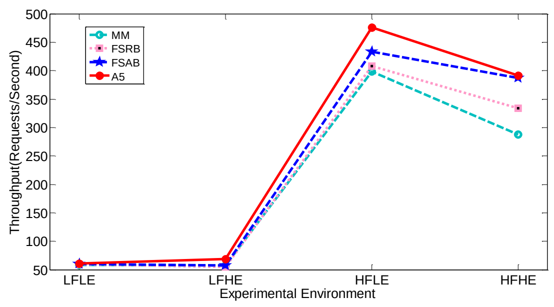
**Figure 8. Fixed (non-adaptive) vs adaptive upper/lower bound**

## B. *Comparison with Other Methods*

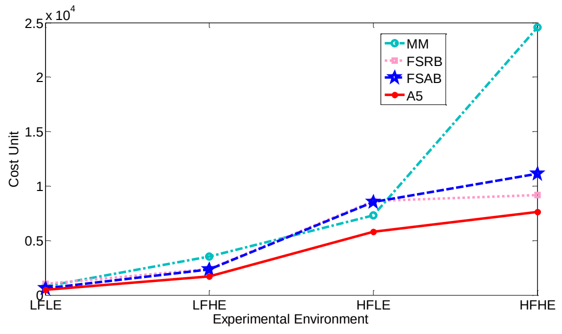
We first compare the A5 algorithm with other methods on the four performance metrics. These methods include traditional Max-Min algorithm (*MM*) with fixed upper/lower-bound, Random Selection of redundant VMs with Fixed upper/lower-Bound (*RSFB*), Random Selection of redundant VMs with Adaptive upper/lower-Bound (*RSAB*). To make the experiments distinguishable, we largely increased the number of workloads in *LFLE* for the sake of making the cost and makespan not always close to zero. This leads to the makespan possibly larger than that in other experimental environments. We consider the priority of performance metrics in an order of throughput, cost and reliability. This leads to a possible lower reliability of A5 than other methods a little bit, which is ignorable. The experimental results are shown in Fig. 9.



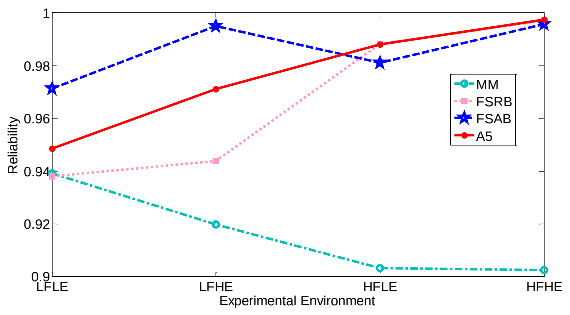
**(a) Comparison over makespan**



**(b) Comparison over throughput**



**(c) Comparison over cost unit**



**(d) Comparison over reliability**

**Figure. 9. Comparison of different methods**

We can conclude the experiments as follows.

(1) In *LFLE*, workloads exert no pressure on the system, thus the performance of each method is similar. Optimization strategy has no noticeable effect when the service capacity is much larger than the impact of the workloads. This advantage of A5 becomes more obvious as the system works in a much higher workload scenario.

(2) Random selection strategy of redundant VMs, which simplify the process, is at the cost of larger makespan, higher cost to schedule the improper redundant VMs and lower throughput. This problem becomes even worse as the workload becomes heavier (with higher arrival rate and longer *EET*). A5 save more than 70% cost and nearly 36.8% of makespan than that of MM. Compared with other methods, the number is not so much, but the improvement is very clear in all the figures.

(3) Adaptive upper/lower-bound selection is better than fixed choice. We can conclude that from Fig. 8 and Fig. 9. When this adaptive bound is combined with a proper selection of the redundant resources (A5), the effects are even better.

## VI. Conclusion and Future Work

In this paper, we proposed A5, An Adaptive Accessing Aware Algorithm to improve throughput and reliability as well as reduce makespan and scheduling cost of a Virtualized Cloud platform consists of many virtual machine resources. The adaptivity is shown not only in upper/lower-bound selection adaptively to decide when to schedule in/out redundant VMs list, but also in Pareto-front based redundant VMs selection adaptively to decide how to execute that schedule. Abundant experiments show that this method outperform other strategies in the above performance metrics.

In future research, we will consider investigating different patterns of arrival rate of workloads with uniform, exponential and normal distributions. Also we will consider the scenario of using predictive methods to ensure even better performance. Last but not least, we intend to implement this algorithm into the real data centers in Tsinghua University with 128 nodes, which will be able to provide sufficient service capacity to validate the effectiveness of this algorithm.

## Acknowledgement

This work is supported by National Science Foundation of China (grant No. 60803017) and Ministry of Science and Technology of China under National 973 Basic Research Program (grants No. 2011CB302505 and No. 2011CB302805) and National 863 High-tech R&D Program (grants No. 2008AA01Z118 and No. 2008BAH32B03). We thank IBM for 2010-2011 IBM Ph.D. Fellowship Award.

## References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, et, al. *Above the clouds: A Berkeley View of Cloud Computing*, Univ. of California, Berkerley, Berkerley, CA, Technical Report No. UCB/EECS-2009-28, 2009.

[2] Y. Zhou, P. M. Chen, and K. Li, “Fast cluster failover using virtual memory-mapped communication”, in *Proc. SCXY’99*, 1999, Rhodes, Greece, pp. 373-382.

[3] G. Levitin, A. Lisnianski, H. Ben-Haim, and D. Elmakis, “Redundancy optimization for series-parallel multi-state systems”, *IEEE Transactions on Reliability*, vol. 47, pp. 165–172, Jun. 1998.

[4] The Hadoop project website. [Online]. Available: http://hadoop.apache.org/core/

[5] G. Sanjay, G. Howard, and L.Shun-Tak, “The Google File System”, *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 29–43, 2003.

[6] C. Ebeling, *An introduction to reliability and maintainability engineering*, New York: McGraw-Hill Science/Engineering/Math, 1996.

[7] W. Kuo, C. L. Hwang, and F. A. Tillman, “A note on heuristic method for in optimal system reliability”, *IEEE Transactions on Reliability*, vol. 27, pp320-324, 1978.

[8] W. Kuo and V. R. Prasad, “An annotated overview of system-reliability optimization”, *IEEE Transactions on Reliability*, Vol. 49, pp.176–191, 2000.

[9] W. Kuo, V., R. Prasad, F. A. Tillman, and C. L. Hwang, *Optimal Reliability Design: Fundamentals and Application*, Cambridge: Cambridge University Press, 2001.

[10] H. Lee, W. Kuon, and C. Ha, “Comparison of Max-Min Approach and NN Method for Reliability Optimization of Series-parellel System”, *Journal of Systems Science and Systems Engineering*. vol. 12, pp39-48, 2003.

[11] A. Azaron, H. Katagiri, K. Kato, and M. Sakawa. “A multi-objective discrete reliability optimization problem for dissimilar-unit standby systems”. *OR Spectrum*. vol. 29, pp235-257, 2007.

[12] D. Salazar, C. M. Roccob, and B. J. Galvan, “Optimization of constrained multiple-objective reliability problems using evolutionary algorithms”, *Reliability Engineering and System Safety*, vol. 91 pp. 1057–1070, 2006.

[13] P. Flocchini, A. Nayak, and M. Xie, “Enhancing Peer-to-Peer Systems Through Redundancy”, *IEEE Journal on Selected Areas in Communications*, vol. 25, 2007

[14] D. W. Coit, “Maxmization of system reliability with a choice of redundancy strategies”. *IIE Transactions*. vol. 35, pp 535-543, 2003.

[15] B. Hong and V. K. Prasanna, “Bandwidth-Aware Resource Allocation for Computing Independent Tasks in Heterogeneous Computing Systems”, in *Proc. PODC’03*, pp. 539-546, Nov. 2003.

[16] B. Hong and V. K. Prasanna, “Distributed Adaptive Task Allocation in Heterogeneous Computing Environments to Maximize Throughput”, in *Proc. IPDPS’04*, pp. 539-546, pp. 52-60, April 2004.

[17] B. Hong and V. K. Prasanna, Adaptive "Allocation of Independent Tasks to Maximize Throughput", *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, pp. 1420-1435, 2007.

[18] C. Zhao, J. Cao, H. Wu, and F. Zhang. Cost Estimation of Advance Reservations over Queued Jobs: a Quantitative Study. International Journal on Modeling, Simulation, and Scientific Computing, Vol. 1, No. 3, 317-332. 2010.

[19] F. Zhang, J. Cao, L. Liu, and C. Wu. Fast Autotuning Configurations of Parameters in Distributed Computing Systems Using Ordinal Optimization. Proc. 38th International Conference on Parallel Processing Workshops, Vienna, Austria, 190-197, 2009.

[20] F. Zhang, J. Cao, L. Liu, and C. Wu. Qualification Evaluation in Virtual Organizations Based on Fuzzy Analytic Hierarchy Process. Proc. 7th Int. Conf. on Grid and Cooperative Computing, Shenzhen, China, 539-547, 2008.

[21] J. Cao, S. A. Jarvis, S. Saini and G. R. Nudd. GridFlow: Workflow Management for Grid Computing. Proc. 3rd IEEE/ACM Int. Symp. on Cluster Computing and the Grid, Tokyo, Japan, 198-205, 2003.

[22] J. E. Rramirez-Marquez, D. W. Coit, and A. Konak, "Redundancy Allocation for Series-parallel Systems Using a Max-Min Approach". *IIE Transactions*. vol. 36, pp.891–898, 2004.

[23] L. Kleinrock, *Queueing Systems. Volume 1: Theory*, John Wiley and Sons, 1975.

[24] X. Yang and G. d. Veciana, "Service Capacity of Peer to Peer Networks", in *Proc. INFOCOM'04*, pp 2242- 2252, 2004.

[25] W. Zhang, J. Cao, Y. Zhong, L Liu, and C. Wu, "An integrated resource management and scheduling system for grid data streaming applications", in *Proc. GRID'08*, pp. 258-265, 2008