Transition Probability-based Resource Selection Algorithm

Author(author@vmware.com)

January 14, 2014

1 Abstract

Hadoop implements a Round-Robin algorithm for its local disks allocation to balance their I/O loading. But when a cluster deployed on virtualization environment, a physical host may place multiple virtual nodes, and a physical disk is mapped by multiple virtual disks. In this case, disk contention on physical layer is hard to control because each virtual disk is scheduled separately. This paper provides a solution to enhance virtual disks allocation policy of Hadoop under virtualzation environment. And focus on the resource selection algorithm about scheduling multiple resource candidates with different capcacity. The simulation result demonstrate the effectiveness of our algorithm, and the compute complexity analysis shows it is able to be applied to production environment.

2 Description

Hadoop has become very widespread across industries, especially in web companies. For each node's disks, Hadoop implements a Round-Robin scheme for disk allocation, this algorithm allocates the candidate disks one by one for every incoming disk I/O request. The purpose of this design is to balance the I/O loading of local disks, and is used by both MapReduce framework and HDFS management. But if Hadoop cluster is deployed on virtualization environment, every node is a virtual machine(VM), the disk I/O occurs on VM's disks is actually mapped to physical host's disks rather than interact with hardware directly.

Consider table 1, for physical host pHost1, VM_1 and VM_2 are hosted on a same physical host, VM_1's disks are created from physical host's physical disk 1 and physical disk 2, VM_2's are from physical disk 2 and physical disk 3. The I/O throughput on each virtual disk of a VM is ultimately redirected to the corresponding physical disk. If the I/O load across virtual disks of each VM is in balance, and all VMs are fairly scheduled. Then the physical I/O load on physical disk 2 will be heavier than that on physical disk 1 and physical disk 3.

To overcome this limitation, we propose a solution which can be summized as 3 steps:

Table 1: Virtual Disks to Physical Disks mapping topology

pHost1	VM_1	VM_2
	$vDisk_{11} \mapsto pHost1 : pDisk1$	$vDisk_{21} \mapsto pHost1 : pDisk2$
	$vDisk_{12} \mapsto pHost1 : pDisk2$	$vDisk_{22} \mapsto pHost1 : pDisk3$
pHost2	VM_3	VM_4
	$vDisk_{31} \mapsto pHost2 : pDisk1$	$vDisk_{41} \mapsto pHost2 : pDisk1$
	$vDisk_{32} \mapsto pHost2 : pDisk2$	
	$vDisk_{33} \mapsto pHost2 : pDisk3$	

Table 2: Disk Mapping Topology of a Given Physical Host

	$pDisk_1$	$pDisk_2$	$pDisk_3$	...	$pDisk_m$
VM_1	a_{11}	a_{12}			
VM_2		a_{22}	a_{23}		a_{2m}
VM_3	a_{31}	a_{32}			
...					
VM_n	a_{n1}		a_{n3}		a_{nm}

- For each physical host, collect its disk topology which maps each VM's virtual disks to this host's physical disks.
- Calculate the disk allocation probability distribution for each VM, section 3 provides a common algorithm for this problem, since this algorithm is straight forward and can find the exact root, we do not provide simulation result.
- Determine disk allocation order based on probability distribution, this step is described in section 4 and simulated in section 5, and it is the core value of this paper.

3 Calculate Disk Allocation Probability Distribution

Take table 2 as example, in this case, $[VM_1, VM_2, ..., VM_n]$ are hosted on a physical host which contains m physical disks($pDisk_1, pDisk_2, ..., pDisk_m$), a_{ij} is the VM_i's allocation probability for its virtual disk which maps to $pDisk_j$.

Our target is to balance I/O loading among physical disks, this can be descriped as an simple optimization issue with linear constraints:

$$\begin{aligned} min \ & \sum_{j=1}^{m} (\sum_{i=1}^{n} a_{ij} - \frac{n}{m})^2 \\ subject\ to \ & \sum_{j=1}^{m} a_{ij} = 1, \ \forall i \\ & a_{ij} = 0, \ \forall (i,j) \notin \Omega \\ & 0 \leq a_{ij} \leq 1, \ \forall (i,j) \in \Omega \end{aligned} \tag{1}$$

Here, Ω is defined as the union of VMs and physical disks pairs. Let M_i be the number of physical disks VM_i occupies. Then a iterative algorithm to calculate the the root of this equation 1

is descriped by algorithm 1.

Algorithm 1: calculate disk allocation probability distribution

Data: disk mapping topology of a given VM
Result: the allocation probability distribution of each virtual disk
initialization: let $K = 1000, \varepsilon = 10^{-4}$;
foreach $i \in (1,n)$ **do** $a_{ij_1} = a_{ij_2} = ... = a_{ij_{M_i}} = \frac{1}{M_i}$;
for $t \leftarrow 0$ **to** K **do**
  $done = 1$;
  foreach $j \in [1,m]$ **do** $S_j = \sum_{i=1}^{n} a_{ij}$;
  foreach $j \in [1,m]$ **do** **if** $|S_j - \frac{n}{m}| > \varepsilon$ **then**
    $done = 0$;
    break;
  else
    continue;
  end
  if $done == 1$ **then**
    return matrix **a**;
  else
    adjust probability values to approximate optimal values;
    foreach $a_{ij}, i \in (1,n), j \in (1,m)$ **do** $a_{ij} = a_{ij} * \frac{n}{mS_j}$;
    foreach $i \in (1,n)$ **do** $T_i = \sum_{j=1}^{m} a_{ij}$;
    foreach $a_{ij}, i \in (1,n), j \in (1,m)$ **do** $a_{ij} = \frac{a_{ij}}{T_i}$;
  end
end

4 Transition Probability-based Resource Selection

Given a VM and its steady-state allocation probability for virtual disks $\mathbf{A} = [a_1, a_2, ..., a_n]$, where n is **A**'s virutual disks number. We need an algorithm to select a virtual disk for each incoming I/O request in this section.

4.1 Design Target

Our target is to balance the I/O loading at real-time, this means any length of sequence generated by this algorithm should approximate to **A**.

For example, assume steady-state disks allocation probability is $[0.1, 0.2, 0.3, 0.4]$, then no matter request for 5, 10, 20 or 200 times, the generated sequence determined by our algorithm should approximate to $[0.1, 0.2, 0.3, 0.4]$.

4.2 Algorithms Description

To acheive this target, our solution is based on transition probability matrix, assume the probability distribution for last request is

$$\mathbf{A}^k = [a_1^k, a_2^k, ..., a_n^k] \tag{2}$$

and there is a transition matrix $\mathbf{P}_A$ given, then the probability distribution of current request is

$$\mathbf{A}^{k+1} = \mathbf{A}^k \mathbf{P}_A \tag{3}$$

We need to customize $\mathbf{P}_A$ to get better performance, firstly, define steplength $d_{k,k+1}$ as:

$$d_{k,k+1} = |(index_{k+1} - index_k) \ mod \ n| \tag{4}$$

In this equation, $index_k$ defined as the selected disk's index number corresponding to $\mathbf{A}$ for the kth request. Generally, our idea is to minimize the these steplengths $[d_{0,1}, d_{1,2}, ..., d_{k,k+1}, ...]$. By doing this, the selection algorithm will tend to select virtual disk one by one to guarantee the disk balance even for very limited times of request.

If we consider Round-Robin algorithm from the perspective of transition probabilty, its steplength is always 1 actually.

Then, for transition probability matrix

$$\mathbf{P}_A = \begin{pmatrix} 0 & p_{1,2} & p_{1,3} & \cdots & p_{1,n} \\ p_{2,1} & 0 & p_{2,3} & \cdots & p_{2,n} \\ p_{3,1} & p_{3,2} & 0 & \cdots & p_{3,n} \\ \cdots & \cdots & \cdots & \cdots & \\ p_{n,1} & p_{n,2} & p_{n,3} & \cdots & 0 \end{pmatrix} \tag{5}$$

If virtual disk i is selected last time, this time virtual disk j will be selected by probability p_{ij}. Minimizing steplengths equals to equation 6, β_j actually reperents the number of non-zero elements of jth column of $\mathbf{P}_A$.

$$\begin{aligned} &min\ \beta_j, \forall j \\ &s.t\ a_j = a_{j-1} * p_{j-1,j} + a_{j-2} * p_{j-2,j} + ... + a_{j-\beta} * p_{j,\beta_j} \end{aligned} \tag{6}$$

The first thing is realigning $\mathbf{A}$ to make sure the adjacent elements do not different too much. Asending sort $\mathbf{A}$ as $\bar{\mathbf{A}} = [\bar{a_1}, \bar{a_2}, ..., \bar{a_n}]$. if n is odd, let $\mathbf{B} = [\bar{a_1}, \bar{a_3}, \bar{a_5}, ..., a_{\bar{N}-1}, \bar{a_N}, a_{\bar{N}-2}, a_{\bar{N}-4}, ..., \bar{a_4}, \bar{a_2}]$, else, $\mathbf{B} = [\bar{a_1}, \bar{a_3}, \bar{a_5}, ..., a_{\bar{N}-2}, \bar{a_N}, a_{\bar{N}-1}, a_{\bar{N}-3}, ..., \bar{a_4}, \bar{a_2}]$

Then let $\mathbf{P}_B$ be the transition probability matrix for $\mathbf{B}$, initialize β_j for each b_j by algorithm 2:

Algorithm 2: Determine no-zero elements number for each column of $\mathbf{P}_B$

```
Data: realigned steady-state probability distribution B
Result: no-zero elements number β_j for each column of P_B
foreach j ← 1 to n do  sum = 0;
for t ← 1 to n do
    sum+ = b_{i−t};
    if sum ≥ b_j then
        β_j = t;
        break;
    end
end
```

Then, our problem can be concluded as optimizing $\widetilde{\mathbf{P}_B}$ under linear constraint, as described by equation 7, an interative algorithm is provided by 3

$$\begin{aligned} &min||\mathbf{B}\hat{P}_B - \mathbf{B}||_2 \\ &s.t. \sum_{j=1}^{m} p_{ij} = 1, \forall i \\ &p_{ij} = 0, \forall (i,j) \notin \Omega_{\hat{P}_B} \\ &0 \le p_{ij} \le 1, \forall (i,j) \in \Omega_{\hat{P}_B} \end{aligned} \tag{7}$$

Algorithm 3: Calculate $\widetilde{\mathbf{P}_B}$ of given $\beta_j, \forall j$

Data: $\mathbf{B}$ and $\beta_j, \forall j$
Result: Transition probability matrix $\mathbf{P}_B$
initialization: let $K = 1000, \varepsilon = 10^{-5}$;
for $t \leftarrow 1$ **to** K **do**
  for $i \leftarrow 1$ **to** n **do**
    count number of no-zero elements number M based on all β_j;
    foreach j **do** $p_{ij} = \frac{1}{M}$;
  end
  calculate $\hat{\mathbf{B}} = \mathbf{B}\mathbf{P}_\mathbf{B}$;
  if $\sum_i (\hat{b}_i - b_i)^2 < \varepsilon$ **then**
    return $\mathbf{B}$;
  end
  foreach i **do** $c_i = \frac{\hat{b}_i}{b_i}$;
  foreach i, j **do** $p_{ij} = \frac{p_{ij}}{c_i}$
end

In some cases, with the input $\beta_j, \forall j$, algorithm 2 is not be able to convergent to the given ε(set to 10^{-5}), to solve it, a greedy stragety is designed as algorithm 4. Since $\mathbf{P}_B \to \mathbf{I}$ can satify $||\mathbf{B}\hat{P}_B - \mathbf{B}||_2 < \varepsilon, \forall \varepsilon$, algorithm 4 will not run forever.

Algorithm 4: Greedy algorithm to convergent to ε always

while *True* **do**
  run algorithm 3; **if** *cannot achieve* $\sum_i (\hat{b}_i - b_i)^2 < \varepsilon$ **then**
    save $\widetilde{\mathbf{P}_\mathbf{B}}$ calculated by 3;
    find $k_{min} = \{k | \beta_j, k^{'} > \beta_j, k_{min}, \forall k^{'}\}$;
    if $\beta_j[k_{min}] == len(\mathbf{B})$ **then**
      break
    else
      $\beta_j[k_{min}] += 1$;
      re-initialize a new $\mathbf{P}_\mathbf{B}$ based on new β_j;
      $\mathbf{P}_\mathbf{B} = 0.5 * (\mathbf{P}_\mathbf{B} + \widetilde{\mathbf{P}_\mathbf{B}})$
    end
  end
end

Finally, let $e_i = (0, 0, ..., 1, 0, ..., 0)^T$, where the i element equals 1, all others equal 0, then the

resource allocation policy for each request can be described as algorithm 5.

Algorithm 5: Algorithm 1

Data: steady-state probability distrubution $\mathbf{B}$, constant $\eta \in [0,1]$
Result: sequence $[\gamma_k]$
initialization: $\alpha = random(n)$, $\pi_0 = e_\alpha$;
calculate transition probability matrix $\mathbf{P}$ of $\mathbf{B}$;
for *kth request from client,* $k >= 1$ **do**
  init selected resource $\gamma_k = 0$ and $S = 0$;
  calculate $\pi_k = \pi_{k-1}\mathbf{P}$ by following algorithm 4;
  random a float value $\phi \in [0,1]$;
  for $t \leftarrow 1$ **to** n **do**
    $S += \pi_k[t]$;
    if $sum \geq \phi$ **then**
      $\gamma_k = t$ and output γ_k;
      break;
    end
  end
  adjust $\pi_k = (1-\eta)\pi_k + \eta * e_{\gamma_k}$;
end

Based on this algorithm, we can guarantee the generated resource sequence approximate to the steady-state probability distribution, no matter how long the sequence is. In another word, select given resources in balance at real-time. If all the probability values of the steady-state distribution are the same, this algorithm is equivalent to Round-Robin algorithm.

4.3 Compute Complexity Anylisis

Consider algorithm 4 and 3, the compute complexity of this algorithm is $o(RKn^2)$, where R is the loop times of algorithm 4, K is the loop times mentioned in algorithm 3, and n is the number of resource candidates.

The maximum value R is n^2 in worst cases, and is determined by ε set by algorithm 3, our experiments in section 5 shows set ε to 10^{-4} or 10^{-5} is enough and R would be much less than n^2.

For parameter K, in algorithm 3 we set it to 1000 for insurance. In our experiments, the declining speed of $\sum_i (\hat{b_i} - b_i)^2$ would be very very slow after adjusted for dozens of times, which means most loops of this step can be skipped if the implementation is awaring of declining speed.

Furthermore, this calculation of transition probability matrix is only need to be executed at the initialization of the resource allocator or when resource candidates changed. So generally, the algorithm itself will not involve much compute resource and can be applied to production environment.

5 Experiment

5.1 evaluation standard

5.2 similarity Definition

Based on cos similarity function to compare two vectors

$$S(\mathbf{B}, \hat{\mathbf{B}_k}) = \frac{\sum b_i * \hat{b_{ki}}}{\sqrt{(\sum b_i^2)(\sum \hat{b_{ki}}^2)}} \tag{8}$$

5.3 Best similarity of a given sequence number

This is for comparison(TODO: add an alg description, and a explaination why this kind of alg is cannot be applied)

Table 3: parameters and their definition

parameter	definition
N	resource number
ε	Algorithm 2
η	Algorithm 5
M	fragment size

Parameters to be used in this section are listed in table 3:

5.4 compare to random alg

The compare result is show in figure 1

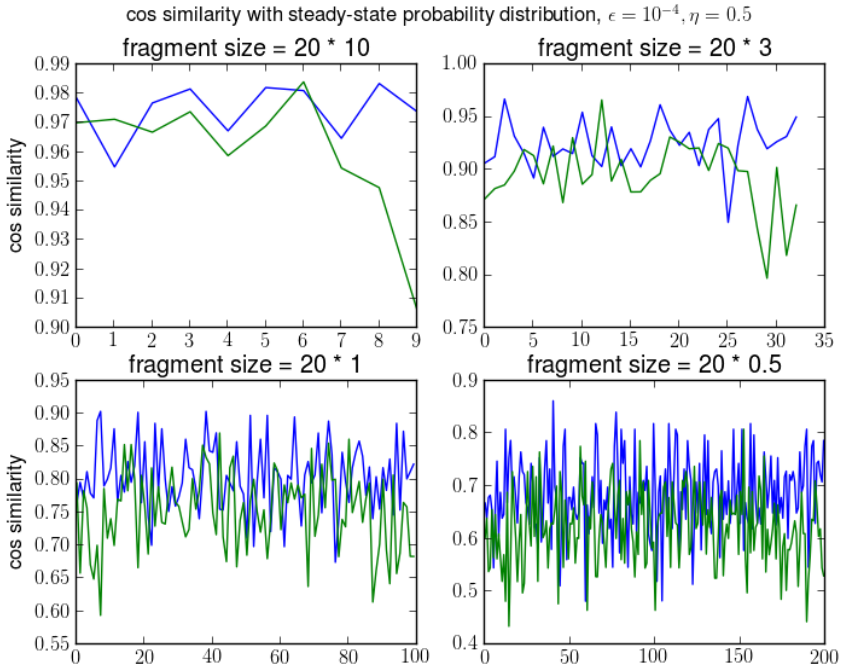

Figure 1: compare01

The mean and mean squared error is shown in figure 2

5.5 test 10 times

In our experiments, the capacity setting for each resource is determined by random, and in section 5.4 is only one test.

In this test, fragment number = 10 and repeat for 10 times. Shown in figure **??**

5.6 test differnt η

In this test, for a given input distribution(so for transition matrix), vary the factor affected by last selection(set to 0, 0.2, 0.4, 0.6, 0.8) to see the different, repeat for 3 times.

Based on this conclusion, we can repeat experiment 5.5 under $\eta = 1$, shown as 5

Figure 2: compare01

5.7 test different ε or step length

consider algorithm 2, ε is the checkpoint to stop looping, the smaller ε, the more calculation, and the larger step-length. step-length is defined as(TODO)...

This test will evalute the influence of ε or step-length, refer to figure 6

6 Apply Scenario

7 Conclusion

To demonstrate the feasilibity of this algorithm, a simulation program is implemented as **git@github.com:xiaodingbian/disk-allocation.git**. In this repository, log file "result_20_40.log" is a running result with 20 VMs and 40 physical disks. And cos similarity function is used to evaluate whether generated sequence for a few requests can follow to steady-state probability distribution.

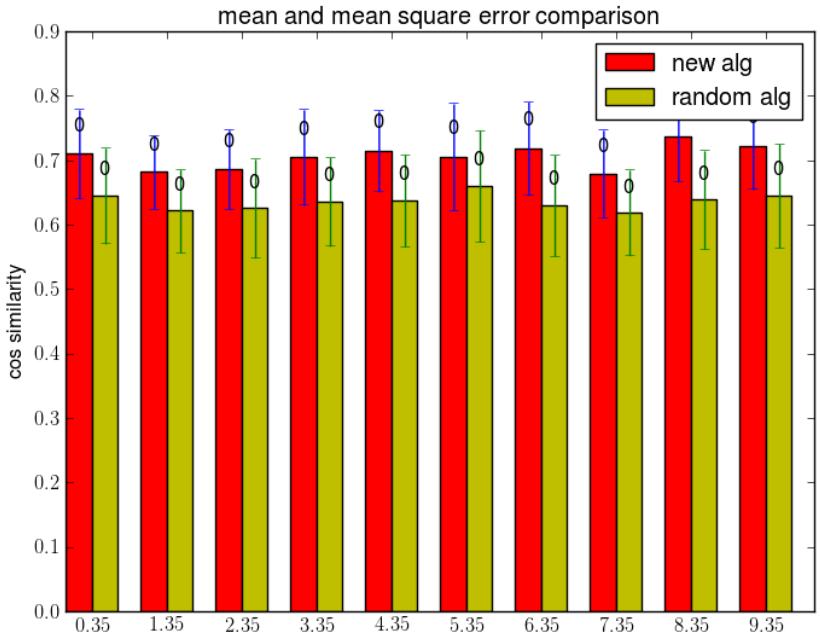

Figure 3: repeat 10 times

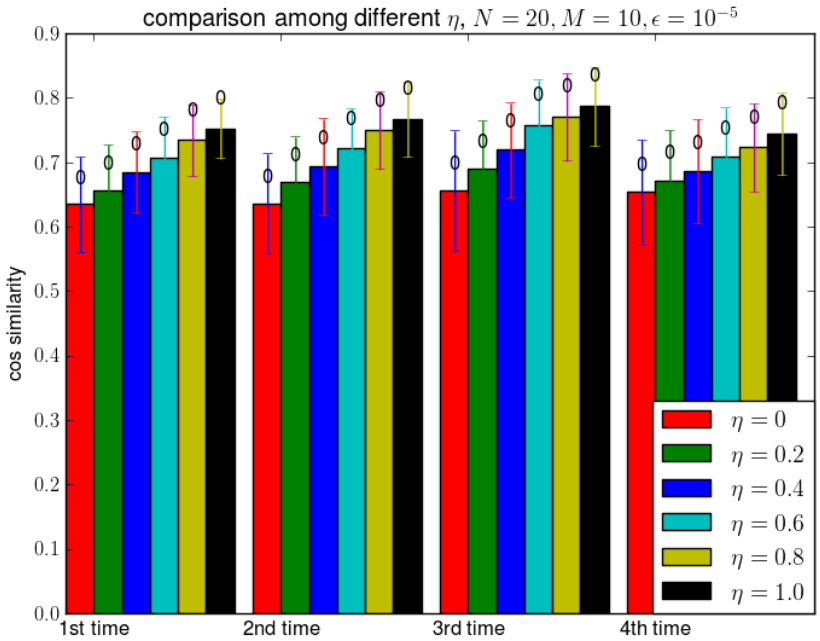

Figure 4: different eta

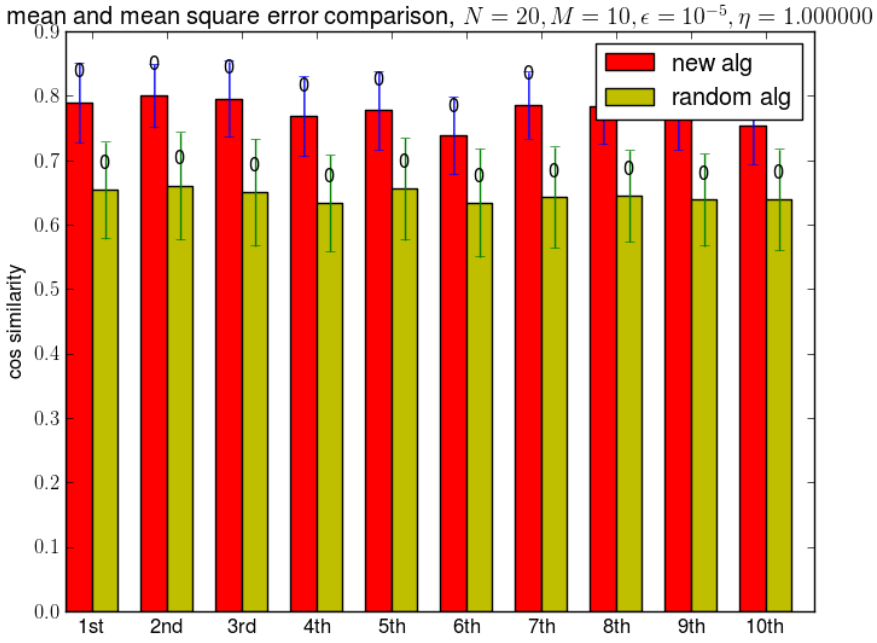

Figure 5: repeat 10 times under $\eta = 1$

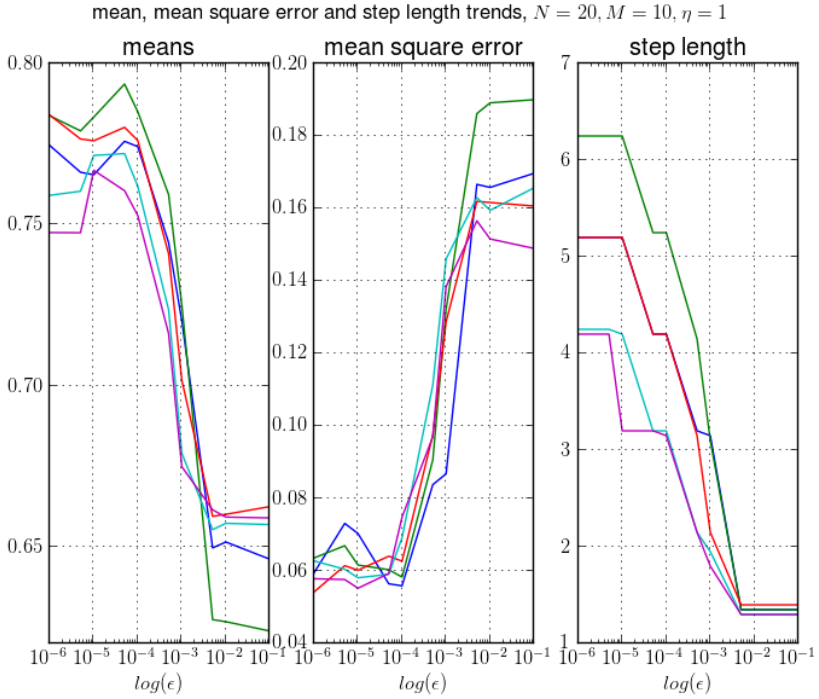

Figure 6: different eta