Scheduling Test Execution of WBEM Applications

Lian Yu, Yu Su
School of Software and Microelectronics
Peking University
P.R. China
lianyu@ss.pku.edu.cn; dongyu1314@hotmail.com

Qianxing Wang
Key Laboratory of High Confidence Software Technologies
Peking University, Ministry of Education,
P.R. China
wqx@pku.edu.cn

Abstract—When test execution of software applications, more often than not testers compete for some testing resources while holding other resources idle. How to optimize utility of testing resources, balance workload of testers and minimize test execution time is the research focus of this paper. We cluster and sequence test cases in order to minimize resource conflicts among testers; and modify the ATC (Apparent Tardiness Cost) algorithm to optimize both global test tasks and individual tester’s tasks. We applied the proposed approach to scheduling test execution of Web-based Enterprise Management (WBEM) applications in a well-known hardware-software vendor and obtained promising results.

Keywords—WBEM, Testing execution, Resource conflicts, Clustering analysis, Sequencing, Scheduling, Setup time.

I. INTRODUCTION

Web-Based Enterprise Management (WBEM) architecture is an initiative of Distributed Management Task Force (DMTF) (www.dmtf.org/standards/wbem/), which aims at providing a framework for industries to build an easy management environment and promoting the information exchange across a variety of underlying technologies and platforms in a consistent format.

WBEM applications administrate large and complex computer systems and network systems, which comprise a diversity of hardware (such as chipset, CPU, memory, hard disk, network card) and software (such as operating systems and application software). Usually, testers face this kind of situations: There are several types of server computers, from high-level to low-level with different capabilities; different types of chipset, CPU, memory, hard disk, network card need to be mounted onto the computers or machines; there are various types of operating systems and application software that need to be installed into the machines; and there are a large amount of test cases to test execute.

To guarantee the quality of WBEM applications, testers have to test every possible configuration of those hardware and software. Due to the large scale and complexity of WEBM applications, testing those WBEM systems incurs tremendous amount of cost.

To reduce the test expense, much effort has been spent on automatically generating test cases and selecting test cases to minimize the numbers based on domain knowledge. This paper is focused on how to improve the testing efficiency during testing execution stage of WBEM. There are three problems of testing WBEM in execution stage:

1) There exist frequent competitions for testing resources among testers, which include bare machines and configuration resources, such as CPU, chipset, hard disk, memory, network card, operation software, and application software. In a real world, the supply of resources is always lower than the demand. Table 1 shows a sample of the supply of resources and the demand of resources by test cases, where the degrees indicate the scarcity of the resources. Lack of overall plan of testing executions, each tester has a variety of test tasks to fulfill, and they tend to request as many of testing resources as possible beforehand, no matter whether they really need them or when to use them.

2) Each tester has several types of computers/machines to perform test execution. In what order to execute those test cases on each machine has significant impact on the testing efficiency, as setup time or configuration time may be very different depending on sequence of test cases executed on that machine. Without any guidance, testers chain test cases on each machine with experience. As a result, the completion time of test execution on each machine becomes longer.

3) Resources for testing execution are always limited, and they have to be circulated among testers. Devoid of global scheduling, scarce resources cannot be optimally utilized. This leads to the testing inefficiency.

Table 1: Resource Supply-demand Degrees

<table>
<thead>
<tr>
<th>Type</th>
<th>Resource Name</th>
<th>Supply</th>
<th>Demand</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon 5405 2.0G</td>
<td>15</td>
<td>57</td>
<td>0.26</td>
</tr>
<tr>
<td></td>
<td>AMD Opteron 8354</td>
<td>5</td>
<td>30</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>Intel S5000VSA</td>
<td>8</td>
<td>114</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>Intel S3200SH</td>
<td>3</td>
<td>97</td>
<td>0.03</td>
</tr>
<tr>
<td>Chipset</td>
<td>Seagate 146G ST3500320NS</td>
<td>12</td>
<td>102</td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td>HighPoint RocketRAID 2640x</td>
<td>5</td>
<td>224</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Scheduling deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives. It has been applied in many fields.
Scheduling is used for computer multitasking and multiprocessing operating system design, and in real-time operating system design in computer science; as a decision-making process for manufacturing and services industries [1].

The problem of this paper can be stated as follows: given a set of testing execution resources, a team of testers, and a set of test cases of a WEBM application where the setup/configuration time between two successive test cases is sequence dependent, perform the testing execution to minimize the make-span (i.e., the last completion of test case execution) and the total tardiness. It is well known that this problem is an NP-hard problem [1]. This paper explores clustering analysis, sort algorithm and a modified job-shop scheduling approach to optimize test execution of WEBM applications given the initial settings as shown in Figure 1:

- Cluster test cases such that test cases in a cluster need similar resources to execute, while test cases in different clusters require different resources. One cluster of test cases is assigned to one tester. By assigning clusters of test cases to testers, it greatly decreases the possibility of resource competitions.
- Queue test cases within a cluster to bare machines that a tester is in charge such that the setup time is minimized from one test case execution to another, and workload on each machine is balanced.
- Dynamically allocate testing resources among different testers as test execution is going on according to tardiness, urgency and feasibility. A scheduling algorithm is proposed to determine the priority of resources among testers whenever resource conflict occurs.

II. CLUSTERING AND SORTING TEST CASES

Clustering analysis, sorting and merging of test cases intend to reduce the competitions of testing resources among testers and search the sequence of test cases within a cluster and minimize the setup time to execute those test cases.

A. Clustering Test Cases

Clustering analysis of test cases categorizes test cases according to similarity of testing resource usage. It makes resource requirements of test cases in same cluster very similar, but quite different among different clusters. Hereafter, we will use the following definitions to describe the algorithms in this paper.

\[ N \]: the total number of test cases
\[ T \]: the total number of testers
\[ R \]: the total types of testing resources, in this paper, there are seven types of resources, including chipset, network card, hard disk, memory, CPU, operating system, and application software.

\( R_{TC}(i) \): indicating that test case \( j \) needs type \( i \) of configuration resource. \( R_{TC} \) is a 7-tuple for the seven types of resources.

- Step 1 - Select cluster centers: Randomly select \( T \) number of test cases as initial cluster centers. Finally each cluster will be assigned to one tester.
- Step 2 - Classify test cases: Calculate the distance \( d \) between each remaining test case \( j \) with test cases in the each cluster center \( i \) \((i=1, \ldots, T)\), pick up the minimal one and put the test case in the corresponding cluster.

\[
\min_{(i,j)} d_{(i,j)} = \sum_{r=1}^{7} || R_{TC}(i) - R_{TC}(j) || \quad (E-1)
\]

Where \( ||R_{TC}(i) - R_{TC}(j)|| \) represents the setup time for the resource \( r \) between test cases \( TC_i \) and \( TC_j \). If two resources used by the two test cases are same, the distance is 0; otherwise the distance is the setup time. There are sequence dependencies among configuration resources. For example, only after have we configured the chipset, we can configure CPU, network card and so on, and only after have we configured the hard disk, we can install the software. Different resources have different configuration costs. Setup time of each type of resources is calculated and estimated by WBEM testing engineers. Table 2 shows the setup time, provided by foremen from a well-known machine vendor in China, for each type of resources.

- Step 3 - Calculate deviation: Compute the average value \( \text{Avg}(i) \) of all distances for each cluster \( i \) \((i=1, \ldots, T)\), and compare the value with the previous iteration average value. If the new value is smaller than the older one, accept the result and update the cluster center. Let \( Ni \) be the number of test cases in a cluster.
\[ \text{Avg}(i) = \left( \frac{\sum_{j=1}^{N_i} d(T_{C_j}, T_{C_i})}{N_i} \right) \]

- Step 4 - Balance test case number for each cluster: Iterate the steps 2 and 3, and finally get a stable clustering result as output. We design a balance mechanism to guarantee the number of test cases in each cluster is close. Define the maximal number of test cases for each cluster as \( G_{\text{max}} \), and count the number of test cases in each cluster during the course of test case clustering, defined as \( G_i \). If \( G_i \) is larger than \( G_{\text{max}} \), then this cluster will be locked and test cases will be considered to bunch into the remaining other clusters.

### Table 2: Resource Setup Time

<table>
<thead>
<tr>
<th>Configuration Resource</th>
<th>Setup Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chipset</td>
<td>6 hours</td>
</tr>
<tr>
<td>Network card and SAN</td>
<td>3.5-4 hours</td>
</tr>
<tr>
<td>Hard disk</td>
<td>2.5-3 hours</td>
</tr>
<tr>
<td>Memory</td>
<td>2.5 hours</td>
</tr>
<tr>
<td>Operating System</td>
<td>2 hours</td>
</tr>
<tr>
<td>CPU</td>
<td>40 minutes</td>
</tr>
<tr>
<td>Application Software</td>
<td>30 minutes</td>
</tr>
</tbody>
</table>

The Clustering approach used in this paper is similar to K-means [2], but has two characteristics: the definition of distance between two test cases in terms of configuration resource requirements; and balance among clusters according to setup time.

### B. Sequencing Test Cases in Clusters

If test cases are sorted in a decent sequence, testers do not have to make much effort in reconfiguring machines when starting new test cases because many resources can be reused. The sorting algorithm consists of the following three steps:

- Step 1 - Compute test case setup time: For each cluster, calculate setup time between each test case pair using the distance \( d \) defined in (E-1). As a result, a test case setup time matrix is generated for each cluster as shown in Figure 2 as an example. Note that the matrix is symmetric.

![Figure 2. Compute setup time matrix](image1)

- Step 2 - Build a min-heap tree [3] for each matrix: use Construct a min-heap tree such that the parent node is always less than the child node as shown in Figure 3 as an example.

![Figure 3. Build a min-heap tree for each matrix](image2)

- Step 3 - Sequence test cases: For each cluster, initially each test case forms a chain, so there are \( N_i \) number of sets for cluster \( i \) at the beginning. We extract the root node of the min-heap tree each time, which contains the current minimal set up time between two test cases, for example node 0.5 between TC2 and TC4 as shown in Figure 4. If these two test cases are contained in two sets, and both of them are located in the head or tail of the set, then merge these two sets; otherwise, abandon the node because at least one test case has already been used in sorting before. For example, node 2.5 contains two test cases, TC1 and TC2, which are at head and tail in different sets, and can be merged. Iterate this step until the set reaches \( M_i \) of the number of bare machines assigned to cluster \( i \).

![Figure 4. Sequence test cases](image3)

- Step 4 - We also design a balance mechanism into the sorting algorithm. \( M_{\text{max}} \) is defined to limit the maximum number of test cases each bare machine can be assigned. If the number of test cases in one bare machine is larger than \( M_{\text{max}} \), the other test cases will be considered sorting in other sets.

### III. SCHEDULING TESTING EXECUTIONS

It is impossible for testers to acquire all testing execution resources up front. Instead, they obtain part of resources, test execute, release the resources, and obtain another part of resource and proceed the testing. That is testing resources need to flow among the testers.

This section discusses the preemption policy, test case execution priority, and resource balance policy.

#### A. Preemption Policy

Preemption policy of scheduling WBEM test execution is closely related to the characteristics of resource
dependency.

To test execute WBEM applications, testers need to configure bare machines with a variety of configuration resources. Chipsets are often used to refer to the core functionality of a motherboard. In this paper, we refer installed chipsets as bare machines, on which hardware CPU, memory, network card and hard disk are mounted, on which in turn operating systems and application software are installed. We construct a resource dependency tree to model the relationship among the resources as shown in Figure 5.

![Resource Dependency Tree](image)

Figure 5. Resource Dependency Tree

The preemption policy will be defined based on the resource dependency tree.

- **No conditional preemption:** A tester is configuring a machine to execute a new test case and requires a new resource which is held by other testers, at least one of whom currently does not use the resource, and will not use till finishing up the current test case. This can be done by scanning the resource dependency trees of other testers. For example, Tester A is to execute a new test case and requires a hard disk, Seagate 146G that is held by Tester B. However, Tester B is using Hitachi 137G instead of Seagate 146G. In this situation, Tester A can no-conditionally preempt the resource from Tester B.

- **Conditional preemption:** A tester is configuring a machine to execute a new test case and requires a new resource which is held by other testers. There exist other testers who do not use the resource now but will use later to execute a test case. Priority will be calculated to determine who can use that resource. For example, Tester A is to execute a new test case and requires a hard disk, Seagate 146G which seized by Tester B. Tester B is currently configuring a CPU, but will configure the same hard disk later on to execute a new test case according the resource dependency tree. In this situation, we start a scheduling algorithm to determine which test case has a higher priority, and allocate the resource to the higher one.

- **No preemption:** A tester is configuring a machine to execute a new test case and requires a new resource which is held by other testers. They have mounted the resource to new test case execution. In this case, no preemption is allowed. For example, Tester A is to execute a new test case and requires a hard disk, Seagate 146G which held by Tester B. Tester B has configured the hard disk to a bare machine and currently test-executing an application. In this situation, the hard disk held by Tester B cannot be preempted till completing the current test case executing by Tester B.

### B. Determining Test Case Priority

Task/job priority is often established according to its processing time, setup time, and slack time [1]. On account of the characteristics of test execution, we propose to take resource usage into consideration. For example, a test case has occupied certain amount of competitive resources, and if we allocate it the rest of resources and let get test execution done, it will release the amount of resources seized, even though its time based priority is restively low.

This paper will determine test case priority based both time and resource occupation.

1) **Time based Priority**

Due to the complexity of the problem, there is no optimum solution to scheduling. We adopt a heuristic approach, and utilize Apparent Tardiness Cost (ATC) algorithm [1] to decide the time based priority for a test case.

ATC approach combines the WSPT rule and the MS rule. WSPT (Weighted Shortest Processing Time first) rule orders test cases in the order of increasing weight/processing times. MS (Minimum Slack time) rule measures the "urgency" of a test case by its slack time.

Under the ATC rule test cases are scheduled one at a time; that is, every time $t$ the machine becomes free a ranking index is computed for each remaining test case. The job with the highest ranking index is then selected to be processed next. The index is defined as

$$ I_i(t) = \frac{w_i}{p_i} \exp \left( \frac{\max (d_i - p_i - t, 0)}{K} \right) $$

(E-2)

Where $p_j$ is the planned processing time for all the remaining test cases of tester $j$; $w_j$ is the weight of each tester, defined by the testing manager; $d_j$ is testing finish time and $\max (d_j - p_j - t, 0)$ is used to describe the minimal slack degree of tester $j$; $\bar{p}$ is the average processing time for all testers; and $K$ is the scaling parameter and can be determined as follows:

$$ R = (d_{\text{max}} - d_{\text{min}}) / C_{\text{max}} $$

Where $(d_{\text{max}} - d_{\text{min}})$ is the difference between the earliest finishing time and the latest finishing time among testers; and $C_{\text{max}}$ is the longest processing time of all testers.

2) **Resource based Priority**

When examining a resource request, we consider three factors: 1) how many other testers who need a resource...
that is no-preemption; 2) how many other testers who have configured or been using a resource that has been configured by a tester; and 3) how many resources are the same between successive test cases for a tester to test execute.

- Currently configured resources: When there is a resource request, the owners of the resource that is no-preemptive will be picked up. For each owner, we will find out the configured resources for its current test cases, and calculate the degree $U_i$ that these resources are required by the other holders. The higher the degree is, the higher the priority value should be. This is because the configured resources cannot be interrupted, and the only way to release them is to let this owner get lacking resources so that he can finish current test case and release the resource in a short time. The $U_i$ is calculated as follows:

$$U_i = \sum_{k=1}^{7} \lambda_k r_k$$

Where $\lambda_k$ is setup time for resource $k$, and $r_k$ is the number of testers who need this resource occupied by the current test case.

- Currently lacking resources: For the holder that is conditional preemptive or no-preemptive, calculate the degree $L_j$ that how many resources except the requested one are further needed to finish the current test case. The higher the degree is, the lower the priority value should be. This is because the higher the lacking resources are, the more difficulty for the holder to get other resource to finish up the current test case and releases the seized resources. The $L_j$ is calculated as follows:

$$L_j = \sum_{k=1}^{7} \mu_k h_k$$

Where $\mu_k$ is setup time for resource $k$, and $h_k$ is the number of testers who have obtained this recourse in the current test case.

- Reusable resources: For the holder of the resource that is conditional preemptive or no-preemptive, calculate the number of reusable resources between successive test cases for a tester to test execute. The larger the number is, the higher the priority value is. Reusable resources between successive test cases save the re-configuration time so that they can finish the current test case and release resources earlier. Therefore, the resource based priority can be calculated as follows:

$$P_j = (U_i + \rho \times R_j)/L_j$$

Where $R_j$ is the number of reusable resources in the next test case for tester $j$; and $\rho$ is weight value, reflecting the importance of reusable resources.

Based on time based priority and resource based priority, we propose an enhanced ATC algorithm as follow:

$$I_i(t) = \frac{w}{p_i} \exp \left( \frac{-\max(d_i - p_i - t, 0)}{kp} \right) \times \alpha$$

$$+ \frac{U_i + \rho \times R_j}{L_j} (1 - \alpha)$$

(E-3)

Where $\alpha$ is the balance factor, ranging from 0 to 1, and it will be further discussed in Section 4.4. In this paper, we call the approach of clustering analysis, sequencing and formula E-2 as enhanced ATC E-ATC; and the approach of clustering analysis, sequencing and formula E-3 as enhanced ATCB E-ATCB.

IV. EXPERIMENTS

This section describes the implementation of the proposed approach, experiment settings and comparison results based on statistic approach.

**A. Implementation of the Prototype**

In order to evaluate the efficiency of the proposed approach in this paper, we develop a prototype of the testing execution scheduling system and perform a variety of experiments on the prototype. Figure 6 shows the overall architecture of the prototype, which consists of two parts: test cases clustering and sequencing, and testing resource scheduling. Test cases clustering and sequencing is responsible for grouping and sorting test cases before executing test cases. Test resource scheduling is responsible for resolving resource conflicts at testing execution stage.

![Figure 6. Architecture of System Prototype](image)

- Test cases XML file: Contain information about all test cases, including basic test case information such as test case ID, Name, and machine resource information (Chipset, CPU, Hard Disk, Memory, Network Card, OS, and Application Software).
- Test cases clustering analysis: Parse test case file, divide test cases into different groups by clustering algorithm, and generate different groups of test cases as output.
- Test cases sorting: For each group of test cases generated in clustering analysis, compute the setup time between each test case pair, create a setup matrix, sort the matrix using min-heap algorithm, and generate a testing execution sequence as output.
- Testing execution: Monitor different tester threads during test execution stage. Once a resource conflict
happens, the tester thread will send a resource request and then get into “waiting” status.

- Conflict listener: Listen to testing execution component, capture resource requests, parse it and invoke scheduling algorithm component.
- Scheduling algorithm: Use scheduling algorithm to compute a resource priority for each resource holder. The tester thread with high priority will get the resource and continue to execute, and the tester thread with low priority will be interrupted and get into “waiting” status.

B. Experiment Settings

To perform the experiments, we visit a real testing execution environment of a WBEM application in a well-known hardware-software vendor, called System Homepage, communicate with the company testing engineers, and collect experiment data with the help of the engineers. Accordingly, we build a simulation environment to investigate the efficiency of the proposed approach by comparing with other solutions, and analyze the impact of parameters.

The following is the settings of the experiment environment: 1) There are 4 testers who are responsible for test execution. 2) There are 16 bare severs. Each tester is assigned 4 bare severs. 3) There are 7 types of testing resources, including chipset, CPU, hard disk, memory, network card, OS, and application software. 4) There are 1,200 test cases that are generated based on 7 types of testing resources.

C. Efficiency Analysis

The first experiment compares testing results in terms of finishing time and total time, among different algorithms. Finishing time indicates the maximum testing days (assume 8-hour per day) among testers, and total time signifies the summation testing days of all testers. The algorithms include FIFS (First in First Served) algorithm, ATCS (Apparent Tardiness Cost with Setups) algorithm, E-ATC (Enhanced-Apparent Tardiness Cost without Balance) algorithm, E-ATCB (Enhanced-Apparent Tardiness Cost with Balance) algorithm.

After performing clustering analysis, 1,200 of test cases are divided into 4 clusters and assigned to the 4 testers as shown in Table 3, where workload is the summation of setup time, hours per tester.

<table>
<thead>
<tr>
<th>Tester</th>
<th>Start date</th>
<th>Finish date</th>
<th>Test Cases</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tester A</td>
<td>2008.06.08</td>
<td>2008.07.02</td>
<td>280</td>
<td>1,604</td>
</tr>
<tr>
<td>Tester B</td>
<td>2008.06.08</td>
<td>2008.07.01</td>
<td>291</td>
<td>1,624</td>
</tr>
<tr>
<td>Tester C</td>
<td>2008.06.08</td>
<td>2008.06.30</td>
<td>317</td>
<td>1,710</td>
</tr>
<tr>
<td>Tester D</td>
<td>2008.06.08</td>
<td>2008.06.28</td>
<td>312</td>
<td>1,660</td>
</tr>
</tbody>
</table>

In order to make the testing comparison results more convincing, we employ t-test of statistic approach to analyze and compare different algorithm results. T-test computes mean, standard deviation of data set and calculates a p-value to indicate difference between two date groups. In statistical hypothesis testing, the p-value is the probability of obtaining a result at least as extreme as the one that was actually observed, assuming that the null hypothesis (no difference between data sets) is true [4].

This paper uses p-value of 0.05 as level of significance which is a standard value to judge the hypothesis. If p-value is less than 0.05, the hypothesis is wrong which means that two group of data has great difference. The lower the p-value is, the less likely the result is, assuming the null hypothesis.

In the experiment, we simulate 30 sets of resource setup time which comply with normal distribution as input, run different algorithms, get 30 testing results for each algorithm, perform normality test on each result set, compare different algorithm results with t-test and finally generate comparison results.

Normality test is necessary before executing t-test, because all data set used for t-test must satisfy normal distribution. Figure 7 shows the normality test result of finishing day dimension, with ATCB algorithm. It is observed that it satisfies normal distribution, and all other algorithms also satisfy normal distribution after the tests.

Table 4 shows the efficiency comparison between the FIFS algorithm and E-ATCB algorithm. It is observed from the table that p-values of both finishing time and total time is 0.000, which means that through clustering, sorting, ATC with balance algorithms, the improvement of efficiency of WBEM is statistically significant compared to the FIFS algorithm.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Algorithm</th>
<th>Mean</th>
<th>StDev</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish Time</td>
<td>FIFS</td>
<td>113.5</td>
<td>275.5</td>
<td>0.000</td>
</tr>
<tr>
<td>Finish Time</td>
<td>E-ATCB</td>
<td>46.9</td>
<td>11.6</td>
<td></td>
</tr>
<tr>
<td>Total Time</td>
<td>FIFS</td>
<td>222.4</td>
<td>53.1</td>
<td></td>
</tr>
<tr>
<td>Total Time</td>
<td>E-ATCB</td>
<td>95.7</td>
<td>21.4</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table 5 shows the efficiency comparison between the ATCS algorithm and the E-ATCB algorithm. In testing execution stage, test case process time includes two parts:
used in the ATCB algorithm, properly deciding values of increasing. The shows the comparison results of testing days as both individual and global testing efficiency. Figure 8 all testers. and the average days indicate the average testing days of maximum testing task finishing days among all testers, ranging from 24 to 35. The finishing days indicate the efficiency, assuring that testers with the most urgent tasks get highest priority. It is observed from Figure 8 that the get highest priority. It is observed from Figure 8 that the difference between two algorithms is that E-ATCB puts in a resource occupation consideration and uses α balance factor to integrate them. E-ATC just focuses on improving the efficiency of individual testers. However, the E-ATCB not only focuses on individual testing efficiency but also global testing efficiency. Therefore, it is observed from Table 6 that although the finishing time of E-ATCB is almost the same as E-ATC, the total time of E-ATCB has been much improved compared to E-ATC, which proves that ATCB is an optimized solution to enhance ATC.

Table 6. Compare between E-ATC and E-ATCB

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Algorithm</th>
<th>Mean</th>
<th>StDev</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish Time</td>
<td>ATCS</td>
<td>93.7</td>
<td>22.8</td>
<td>0.000</td>
</tr>
<tr>
<td>Finish Time</td>
<td>E-ATCB</td>
<td>46.9</td>
<td>11.6</td>
<td>0.000</td>
</tr>
<tr>
<td>Total Time</td>
<td>ATCS</td>
<td>192.6</td>
<td>45.4</td>
<td></td>
</tr>
<tr>
<td>Total Time</td>
<td>E-ATCB</td>
<td>95.7</td>
<td>21.4</td>
<td></td>
</tr>
</tbody>
</table>

Table 6 shows the efficiency comparison between E-ATC algorithm and E-ATCB algorithm. The only difference between two algorithms is that E-ATCB puts in a resource occupation consideration and uses α balance factor to integrate them. The E-ATC just focuses on improving the efficiency of individual testers. However, the E-ATCB not only focuses on individual testing efficiency but also global testing efficiency. Therefore, it is observed from Table 6 that although the finishing time of E-ATCB is almost the same as E-ATC, the total time of E-ATCB has been much improved compared to E-ATC, which proves that ATCB is an optimized solution to enhance ATC.

D. Impact Analysis of α Factor

This experiment analyzes the impact of balance factor α used in the ATCB algorithm, properly deciding values of α will lead to satisfying results that take into consideration both individual and global testing efficiency. Figure 8 shows the comparison results of testing days as α value increases. The X coordinate is α value, ranging from 0 to 1, and the Y coordinate is testing days (8-hour per day), ranging from 24 to 35. The finishing days indicate the maximum testing task finishing days among all testers, and the average days indicate the average testing days of all testers.

If α value is 1, the algorithm is the ATC algorithm, which focuses on improving the individual testing efficiency, assuring that testers with the most urgent tasks get highest priority. It is observed from Figure 8 that the finishing days have decreased as α increases. If α value is 0, the algorithm is resource based algorithm, which focuses on improving the global testing efficiency. It is observed from Figure 8 that the average days have decreased as α decreases. How to properly choose α value to make balance between these two scheduling algorithms is important. From Figure 8, we observe that when the α value is between 0.4 or 0.8, the result is most satisfying in the setting of this experiment.

Figure 8. Testing days with increased α

To improve the testing efficiency much of effort has been put on test case selection and automatically generating test cases [5][6][7]. Due to the characteristics of WBEM applications, the efficiency of testing can also come from proper scheduling of test execution. This paper designs a systematic approach to optimize the scheduling of test executing WBEM applications.

Clustering techniques have been used for different purposes in software testing, for example, test case selection and defect prediction [5][7][8][9][10][11]. This paper uses the technique to schedule test execution. In statistics and machine learning, k-means clustering is a method of cluster analysis which aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean [2]. The Clustering approach used for test case clustering in this paper is similar to K-means, but has two characteristics: 1) the definition of distance between two test cases in terms of configuration resource requirements; and 2) balance among clusters according to setup time.

The first systematic approach to scheduling problems was undertaken in the mid-1950s. Since then, thousands of papers on different scheduling problems have appeared in the literature. The majority of these papers assumed that the setup time (cost) is negligible or part of the job processing time (cost). While this assumption simplifies the analysis and/or reflects certain applications, it adversely affects the solution quality of many applications of scheduling that require an explicit treatment of setup times (costs).

The interest in scheduling problems that treat setup

The scheduling problem with setup time in this paper has different characteristics from those of described above. The setup time is not only test case sequence-dependent, but also resource relation-dependent.

VI. CONCLUSION AND FUTURE WORK

Test executing WBEM applications is a complicated scheduling problem. It has sequence-dependent and resource hierarchy-dependent setup times, and needs to meet multiple objectives, such as makespan and workload balance among testers. This paper takes a systematic approach to assign test cases to testers and dynamically allocate limited testing resources among testers. The proposed approach starts with clustering test cases according to their similarity of test resource requirements such that test cases in a cluster have similar resource requirements, while test cases in different clusters have different resource requirements. This reduces the competition of resources among testers. In a cluster, sort and sequence test cases such that the setup time is minimal. As a result, the makespan is minimized. In addition, we take resource-based priority into consideration, and modify Apparent Tardiness Cost (ATC) such that testing resources are allocated to testers to achieve global balance of resource usage.

We compare our proposed approach with FIFS (first come first serve) and the ATCS, and the results demonstrate the advantage of our proposed approach. We also analyze the impact of global balance of resource usage.

Our future work includes further exploring the algorithms to schedule test cases for test execution, take the skill and preference of testers into consideration, and apply the proposed approach to practical WBEM applications in industry.

ACKNOWLEDGEMENT

The work presented in this article is partly sponsored by IBM Shared University Research Grant and by the National High-Tech Research and Development Plan of China (No. 2006AA01Z175). The authors would thank Bo Hou and Jing Tao Zhao for working on the empirical study described in this paper.

REFERENCES