Scheduling policy for on-line translation service

Małgorzata Sterna · Jacek Juraszek

Abstract The research concerns the scheduling tool supporting the web service offering automatic text translation between Polish and three other languages. The web service accepts client orders submitted via WWW site and realizes them using specialized translation software. The system consists of a few translation processes dedicated to particular translation directions, which run in parallel sharing hardware resources. After the deep analysis of real historical data, obtained for a simple random scheduler, we proposed an optimized on-line scheduling policy for managing client orders, based on a priority dispatching rule. It improved the efficiency of the web service, fulfilling all the requirements specified by the owner of the system.

1 Introduction

In the last decade the number of users of web services has increased significantly, causing problems in ensuring the satisfactory quality of these on-line services. Applying even very simple scheduling techniques may be very helpful in improving the efficiency of such systems. In this paper, we investigate the web service for automatic language translation, provided by one of the Polish publishing companies. It offers six translation directions between a national language and three foreign languages: English, German and Russian. The translation process is performed by the highly specialized software designed by academic experts. It enables translating not only single words but whole sentences too, taking into account grammatical rules of particular languages. The service is provided free of charge via WWW site. Users can submit requests of the length restricted by the owner of the system to 999 characters. Moreover, it is assumed that translation tasks have to be completed within the 180-second time limit. After this deadline a task is canceled and a user receives the information about the failure in the translation process. The owner of the system assumed that at most 0.5% of client orders...
may be non-served for any reason. All user requests submitted via WWW site are
managed by a control procedure, which separates them to different translation processes
based on the required translation direction. Two most popular translation directions
from/to English are served by three identical processes each, while the remaining ones
are associated with only one process. Hence, the web service is realized by 10 trans-
lation processes, which are theoretically independent. However, since these processes
share system resources (Intel Xeon X5355 2.66 GHz computer with 8 processors and
7GB RAM) they influence each other. In practice, the processing speed depends on
the workload of the web service. During the rush hours, when the number of submit-
ted requests increases significantly, a translation task may require more time due to
processors time sharing. Thus, the processing time for a particular translation task is
unknown a priori and it depends on the contents of the user request and, additionally,
on many technical issues such as e.g.: the system workload, runtime errors in the trans-
lation software or communication delays. Hence, we deal with the on-line scheduling
problem, where user orders are not known in advance and they appear dynamically in
the system, whose state changes in the non-deterministic way.

Originally, the control algorithm managing user tasks was based on simple First-
In-First-Out (FIFO) rule (cf. e.g. [1], [6]) and random assignment of English tasks to
translation processes (for the remaining types of tasks only one process is available).
However, the quality of service was unsatisfactory due to too many non-served client
requests. The percentage of non-completed tasks exceeded the 0.5% limit established
by the owner of the service. Within our research, we proposed an optimized control
procedure increasing the quality of the analyzed web service in the extent meeting all
system requirements. The performance of the new scheduling policy was checked in
extended computational experiments, using the historical long-period data collected in
the web service log.

The organization of the paper is as follows. Section 2 presents a formal definition
of the problem under consideration. Section 3 shows the analysis of real values of the
web service parameters, which allowed us to design a new efficient scheduling policy
adjusted to the specificity of the system. The idea of this on-line algorithm is presented
in Section 4. Section 5 contains the main observations resulting from the computational
experiments. The paper is concluded in Section 6.

2 Problem Formulation

The web service under consideration can be modeled as a specific on-line case of the
multi-purpose machine scheduling problem (cf. e.g. [2], [3]). At a certain time moment,
al l user requests for translating texts submitted to the web service form the set of
available tasks \( T = \{T_1, \ldots, T_n\} \), while ten translation processes running within the
system can be modeled as the set of multi-purpose machines \( M = \{M_1, \ldots, M_{10}\} \).
Obviously, particular six translation directions offered by the web service are as-
associated with given subsets of machines (translation processes). Hence, translations
from and to English are served by any machine from set \( E_1 = \{M_1, M_2, M_3\} \) and
\( E_2 = \{M_4, M_5, M_6\} \), respectively. Translations from/to German, can be realized by
only one machine, i.e. \( G_1 = \{M_7\} \) and \( G_2 = \{M_8\} \), respectively. Similarly, user re-
requests concerning translations from/to Russian are executed by only one process, i.e.
\( R_1 = \{M_9\} \) and \( R_2 = \{M_{10}\} \), respectively. The parameters of a task become known
only after submitting a user request to the web service. After the task arrival to the
system it is available for processing. Particular tasks $T_j$, i.e. requests for text translation, are described by the following parameters (assuming that time parameters are given in seconds which passed from the last restart of the system):

- the arrival (release) time: $r_j$,
- the deadline: $d_j$ ($d_j = r_j + 180$),
- the number of characters in the translated text: $c_j$ ($c_j \leq 999$),
- the number of words in the translated text: $w_j$.

Moreover, depending on the translation direction required by a user, we define for task $T_j$ the subset of machines $\mu_j \in \{E_1, E_2, G_1, G_2, R_1, R_2\}$, on which it can be executed.

In general, each task $T_j$ has to be performed by one machine selected from set $\mu_j$ without preemption. However the machines (i.e. translation processes) work in the non-deterministic way and our knowledge about their state is very restricted. Actually, the machines model the processes associated with the specialized translation software, designed by external developers, which still contains some implementation errors. Unfortunately, for unknown reasons, not all tasks finish with success and, due to runtime errors, translation processes may need restarting by the operating system. Consequently, machines (modeling theses processes) may be unavailable in some periods, but these unavailability periods cannot be predicted and even detected. To overcome this difficulty, the processing time limit for a single attempt to task execution was introduced. After exceeding this limit, the execution of a task is considered as unsuccessful. If task $T_j$ is not completed successfully, then its execution is repeated possibly on another machine from set $\mu_j$. The failure in execution may result from: exceeding the time limit, the unavailability of a machine or an internal software error in the translation process. Hence, in general, tasks cannot be preempted, but there might be a few attempts to their completion in the system. This means that the execution of a task can be preempted only due to the technical reasons mentioned before. These multiple attempts to completing a task can be considered as non-resumable preemptions of tasks (cf. e.g. [4], [7]).

From the theoretical point of view, particular translation directions (sets of machines $E_1, E_2, G_1, G_2, R_1, R_2$) might have been considered as separate scheduling problems with 3 parallel machines for English ($E_1, E_2$) and with a single machine for the remaining two languages ($G_1, G_2, R_1, R_2$), because sets $\mu_j$ defined for particular types of tasks (i.e. translation directions) are disjoint. However, we decided to model the web service as the one integral system to reflect the goal of the research, which is to improve the quality of the whole web service. Moreover, as we have mentioned, although particular translation processes, represented by machines, are independent they share hardware resources and, in practice, influence one another. This influence cannot be expressed formally, but it may be observed in the system log data. The huge number of tasks requesting one machine consumes processors time and may slow down other machines. Runtime errors occurring in one translation process may make the system unstable and lead to the failure of other processes. The machines in the formal model correspond to software processes not to physical processors, which those processes share. Of course, it is possible to apply different scheduling strategies for particular types of tasks, but the performance of the proposed policies should be validated with regard to the entire system.

As we have mentioned, the owner of the system imposed two efficiency constraints, which should be obeyed by a scheduling policy: the 180-second time limit for particular client requests and at most 0.5% of requests non-completed by the service within this time limit. Consequently, the main goal of any policy controlling the web service is to
execute as many tasks as possible and to ensure short response time of the system.

To evaluate the quality of the service, we collect a few task parameters, which become available after processing task $T_j$ by the web service, namely:

- the completion time: $C_j$,
- the flow time (the system response time): $F_j = C_j - r_j$,
- the number of trials of task execution: $a_j$,
- the processing time in trial $i$: $p_j(i), i = 1..a_j$,
- the machine used in trial $i$: $M_j(i) \in \mu_j, i = 1..a_j$,
- the status of trial $i$: $s_j(i) \in \{S, T, F, M\}$, $i = 1..a_j$.

If the last trial of executing task $T_j$ finished successfully ($s_j(a_j) = S$), then the processing time $p_j(a_j)$ denotes the translation time for task $T_j$. The unsuccessful execution of a task may result from exceeding the time limit ($s_j(i) = T$), from the failure in translation ($s_j(i) = F$) or from the machine failure, i.e. the runtime error of the translation software ($s_j(i) = M$). The processing time consumed by the unsuccessful trials can be considered as wasted one, since these trials did not lead to the task completion.

Obviously, such trials cannot be usually avoided, because they result from different kinds of failures, which appear in the real Internet service. However a scheduling policy should detect them as early as possible, minimizing the total wasted time, which is an additional parameter reflecting the efficiency of the system, besides the percentage of non-completed tasks and the average response time mentioned above.

3 Web Service Analysis

The translation service under consideration was originally managed by a very simple scheduling policy. It assigned user requests submitted to the system (tasks) to a proper translation process (machine) in a random way (in the case of a few processes/machines which may serve a task). The queues associated with particular translation processes (machines) were handled according to the First-In-First-Out (FIFO) strategy. This simple approach resulted in too low quality of service in terms of the average system response time and, first of all, of the percentage of non-completed user requests.

In order to optimize the scheduling policy, we carefully analyzed the web service log to determine the possible ways of improving the algorithm. This analysis was restricted to a single working day for which one of peak workloads was detected. The selected data set contained 60750 translation requests, i.e. nearly one request submitted every second. Actually, the number of tasks handled by the web service is rather stable over the week, so restricting the analysis to one day is fully justified. The number of tasks present in the system changes significantly during the day (cf. Figure 1). The rush hours are between 15:00 and 19:00, when the number of tasks is nearly 30 times higher than for the hour with the minimum load (6:00-7:00). The numbers of requests submitted for particular translation directions differ very much too. The majority of submitted requests concerns translation from English to Polish (47.91%) and from Polish to English (35.38%), i.e. together 83% of all submissions. These two translation directions are handled by three identical processes each (three identical parallel machines). Requests involving German amount to about 11% (5.8% of German-Polish translations and 5.31% in the opposite direction), while submissions involving Russian amount to only 6% of the total number of tasks (3.7% of Polish-Russian translations and 1.89% in the opposite direction). These translation directions are served by only one process each (single machines). The analysis of the web service log suggested two
Fig. 1 The number of tasks in particular subperiods of the 24-hour sample with 60750 tasks
Table 1 The correlation coefficient for the task processing time and the number of words

<table>
<thead>
<tr>
<th>Translation direction</th>
<th>Correlation coefficient</th>
<th>Average processing time [sec.]</th>
<th>Std. dev. of processing time [sec.]</th>
<th>Average words number</th>
<th>Std. dev. of words number</th>
</tr>
</thead>
<tbody>
<tr>
<td>pl→en</td>
<td>-0.051</td>
<td>4.7</td>
<td>2.4</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>en→pl</td>
<td>-0.039</td>
<td>7.7</td>
<td>6.3</td>
<td>16</td>
<td>28</td>
</tr>
<tr>
<td>pl→ge</td>
<td>-0.004</td>
<td>6.5</td>
<td>6.2</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>ge→pl</td>
<td>0.402</td>
<td>5.2</td>
<td>3.4</td>
<td>12</td>
<td>21</td>
</tr>
<tr>
<td>pl→ru</td>
<td>-0.212</td>
<td>3.9</td>
<td>1.5</td>
<td>9</td>
<td>23</td>
</tr>
<tr>
<td>ru→pl</td>
<td>-0.384</td>
<td>3.1</td>
<td>1.7</td>
<td>17</td>
<td>26</td>
</tr>
</tbody>
</table>

depends more on the state of the runtime environment than on the length of the text. In consequence, the optimized scheduling policy should rather focus on enforcing multiple trials of executing a task, hoping for success, than on scheduling tasks according to their predicted processing times, waiting for their completion in the first attempt.

4 Scheduling Policies

The web service under consideration is built from a few software components designed by different companies. For this reason, the scheduler does not have the full control of the system and it has to take into account the specificity of the translation processes. It accepts user requests and executes tasks associated with them on particular machines (i.e. translation processes). The scheduler manages the queues associated with machines deciding on the order of task executions. Moreover, it supervises a particular task execution, handling possible errors or terminating the executions which exceeded the given time limit. Taking into account two facts that: the task processing time is unknown and it depends rather on the system workload than on the submitted text, as well as that: all tasks have to be completed within the same 180-second limit, the queues associated with machines should be scheduled according to FIFO strategy (i.e. according to task arrival times \( r_j \)). This priority dispatching rule was applied in both scheduling policies: the simple one, used originally in the web service, and in the optimized one, proposed within this research. In consequence, the only way to increase the efficiency of the system was optimizing the algorithm handling the execution of a task. The general idea of this method is presented in Algorithm 1.

First the algorithm validates the deadline for task \( T_j \), i.e. it checks whether less than 180 seconds passed since submitting it to the system at time \( r_j \). This constraint is checked at every attempt to completing task \( T_j \). Particular trials are numbered with \( i \) from 1 to \( \alpha_j \). Variable \( t \) represents the current time in the system counted in seconds from the last restart of the web service. If the deadline for a task is exceeded, its execution fails due to timeout (\( s_j(i) = T \)) and the user receives the information on the failure in text translation. Otherwise the scheduler tries to execute task \( T_j \). First, procedure \( ChooseMachine(T_j, \mu_j, i) \) determines machine \( M_j(i) \) selected from a set of eligible machines \( \mu_j \) which will be used for processing \( T_j \) in trial \( i \). Then procedure \( Translate(T_j, i, M_j(i), p_j^{max}(i)) \) puts \( T_j \) in the queue for \( M_j(i) \) and supervises its execution, guaranteeing that the deadline \( (r_j + 180) \) will not be exceeded. Moreover, it assures that the execution time \( p_j(i) \) will not be longer than the time limit given for the \( i \)-th attempt to the task completion on machine \( M_j(i) \), i.e. than \( p_j^{max}(i) \). If the \( i \)-th attempt finishes successfully (\( s_j(i) = S \)), then the translated text is sent to the user.
Algorithm 1 Serving Translation Request ($T_j$)

\begin{algorithm}
\begin{algorithmic}
\State $i \leftarrow 1$
\Loop
\If{$t \geq r_j + 180$}
\State $s_j(i) \leftarrow T$
\State $j \leftarrow 1$
\EndIf
\State $M_j(i) \leftarrow \text{ChooseMachine}(T_j, \mu_j, i)$
\State $(s_j(i), p_j(i)) \leftarrow \text{Translate}(T_j, i, M_j(i), p_{j, \text{max}}(i))$
\If{$s_j(i) = S$ or $t \geq r_j + 180$}
\State $j \leftarrow 1$
\EndIf
\State $i \leftarrow i + 1$
\EndLoop
\State $a_j \leftarrow i$
\end{algorithmic}
\end{algorithm}

Otherwise, the scheduler encounters the failure in the task completion and it tries to complete it once more in the next $(i+1)$-st attempt.

The simple scheduling policy performed the random selection of a machine from set $\mu_j$ and applied the same processing time limit $p_{j, \text{max}}(i) = 180$ in every trial and for all machines. The simple strategy assumed that machines are reliable and tasks are likely to be completed at the first attempt. The historical data showed that these assumptions were unrealistic. The optimized policy assigns task $T_j$ to the machine with the shortest queue of tasks waiting for execution (if $|\mu_j| > 1$). In the case of failure in task execution, at the next $(i+1)$-st attempt to completing $T_j$, the scheduler eliminates from the analysis machine $M_j(i)$, which was not able to perform the task in the previous $i$-th trial (in the case of one more trial this machine will be available again). The analysis of the web service log showed that failures often result from errors in the translation process, which needs restarting by the operating system. Within this restarting time the process (machine) is non-available, but the scheduler is not informed about the machine state. Shifting tasks to another machine from $\mu_j$ makes its successful execution more likely, because the translation process represented by another machine may be normally working (not being restarted) and may be executed by another processor with lower workload.

In order to detect the machine non-availability, the optimized scheduling policy applies the much shorter time limit for a single attempt to task execution than the simple strategy. Moreover, this time limit is not constant, but it increases with the trial number. In this way, the scheduling policy allows for translating some specific texts, which are linguistically correct, but require the longer processing time than on average. Additionally, the time limit is determined separately for particular machines (processes) in the system, based on the data collected in the web service log. Consequently, the time limit is not common for the whole web service, but it has been tuned for particular translation directions separately, reflecting their efficiency, reliability, error resistance etc. Summing up, the processing time limit for the $i$-th attempt to completing task $T_j$ on machine $M_j(i)$ is determined as: $p_{j, \text{max}}(i) = \min\{P95\%_{M_j(i)} + (i-1) \cdot SD_{M_j(i)}, 180 - \sum_{k=1}^{i-1} p_j(k)\}$, where $P95\%_{M_j(i)}$ denotes 95-th percentile of the processing time for successful translations on machine $M_j(i)$ (i.e. the upper bound of the processing time for 95% of tasks executed on this machine based on the web service log), $SD_{M_j(i)}$ denotes the standard deviation of the processing time for successful translations on
machine $M_j(i)$. The second factor corresponds to the time limit decreased with the durations of previous trials for this task (if any).

5 Computational Experiments

The optimized scheduling policy was compared to the simple scheduler in the computational experiments performed for real data extracted from the web service log described in Section 3. The system log was used as input data for the new algorithm. Hence, both strategies were evaluated for the same set of 60750 tasks submitted by the users of the web service within 24-hour period. Both policies received the same stream of user requests in terms of their contents and arrival times. For each task its execution history was logged including its final status (determined by the result returned by the translation algorithm or by the decision of the scheduler).

The optimized policy increased significantly the efficiency of the service and met all requirements defined for the system. The simple strategy resulted in 0.57% of non-completed tasks. This level was unacceptable due to the 0.5%-limit imposed by the owner of the web service. The optimized algorithm decreased the percentage of unsuccessful translations more than 3 times to only 0.18%. However, these values, aggregated for the whole 24-hour period, do not reflect the dynamics of the web service. The percentage of faults in task executions in particular subperiods (cf. Figure 2) showed that even at rush hours the optimized policy did not exceed the 0.5% limit. For example, in period 15:00-16:00, it led to less than 0.2% of faults, while the simple algorithm failed in executing nearly 2% of tasks. For the heavy workload of the web service, the system efficiency depends strongly on the scheduling policy, because the queues associated with particular machines become longer and exceeding the deadline for particular tasks is more likely. In such case, the execution of tasks must be optimized - the simple strategy is definitely not sufficient. When the number of tasks submitted to the system is smaller, then both algorithms behave similarly, since faults in tasks completions result mainly from the translation failures, which cannot be eliminated - they will appear for any scheduling policy, because of e.g. grammatical mistakes in the submitted texts.

![Fig. 2 Percentage of non-executed tasks in particular subperiods from 60750 tasks](image-url)
Table 2 The efficiency factors for particular translation directions for both policies

<table>
<thead>
<tr>
<th>Translation direction</th>
<th>Percentage of faults [%]</th>
<th>Average flow time [sec.]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>simple policy</td>
<td>optimized policy</td>
</tr>
<tr>
<td>pl→en</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>en→pl</td>
<td>0.83</td>
<td>0.03</td>
</tr>
<tr>
<td>pl→ge</td>
<td>3.07</td>
<td>3.04</td>
</tr>
<tr>
<td>ge→pl</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>pl→ru</td>
<td>0.18</td>
<td>0.18</td>
</tr>
<tr>
<td>ru→pl</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

This observation is confirmed by the analysis of the percentage of faults encountered for particular translation directions (cf. Table 2). The software modules designed for particular translation directions differ in their quality. The high percentage of faults in translations from Polish to German resulted from the low quality of the software. The number of tasks associated with this service was relatively small (about 5%), therefore the system could have executed all of them without exceeding time limit. The failure in execution were caused by translation errors, independent of the scheduling policy. For this reason, the improvement observed for the optimized strategy was insignificant. But for the service translating from English to Polish, for which the number of submitted tasks was the largest one, the percentage of faults decreased from 0.83% to 0.03%. In this case, the quality of translating software was high and faults in task executions resulted mainly from exceeding the deadline.

![Fig. 3](image)

Fig. 3 The average task flow time in particular subperiods with the standard deviation values

The optimized scheduling policy not only decreased the number of non-completed tasks, but it also shortened the response time of the system from 9.1 to 4.1 seconds on average, i.e. the new strategy reduced the time the users waited for text translation more than twice. Moreover, the system behavior became very stable. The analysis of the average task flow time (i.e. the system response time $F_j$) in particular subperiods (cf. Figure 3) disclosed the weakness of the simple policy. The response time grew with the system workload. In rush hours, users had to wait for text translation nearly three
times as long as in the night. Furthermore the response time was unpredictable, which is reflected in the huge standard deviation values. The optimized policy ensured the stable response time independently of the system workload. The flow time reduction (and the stabilization of its value) is also visible for particular translation directions (cf. Table 2), especially for the most occupied service translating from English to Polish. For this service the process of scheduling tasks became crucial due to the large number of user requests present in the system at the same time.

The improvement of the quality of the web service, visible in the decreased percentage of non-executed tasks and the shortened average response time, resulted mainly from changing the scheduling policy by: imposing simple load-balancing among parallel machines and enforcing multiple trials of task executions in case of failures. These changes in the strategy influenced also other parameters describing the efficiency of the system. They caused the reduction in the average processing time of tasks (from 6.2 to 3.6 seconds) and the time spent by the web service on unsuccessful text translations (from 7.29% to 4.44% of the total CPU time). Moreover, the new strategy eliminated the failures resulting from the system errors caused by overloading it and decreased the number of failures resulting from exceeding the time limit.

6 Conclusions

The paper presented the on-line algorithm for scheduling tasks in the real web service for text translation. The efficiency of the existing simple strategy was not sufficient, since it did not meet the requirements specified by the owner of the system. The Internet service was formally described as the complex scheduling environment with two separate sets of parallel identical machines and four single machines, corresponding to different translation directions. Although tasks submitted for particular types of translations are scheduled separately, they share the hardware resources and they had to be analyzed simultaneously. The schedule has to be constructed on-line in real-time, since the set of tasks is unknown in advance and unexpected events appear in the system. We proposed the list scheduling algorithm, which handles tasks submitted to the web service and fully collaborates with the specialized translation software.

The computational experiments showed that the optimized policy increased the efficiency of the web service. It fulfilled all requirements specified by the system owner, which had not been satisfied by the simple policy used so far. Thus, the main goal of the research was reached. The optimized strategy replaced the previous one in the web service. Moreover, the analysis of test results showed some weak points of the system, independent of the scheduling policy, such as the low quality of some modules of translation software, as well as the lack of the mechanism detecting failures in translation processes.

Due to the increasing number of the web service users the system under consideration has been recently enlarged by changing its architecture and adding additional hardware and software resources. Within the future research we will have to propose a new formal model of the web service and to design scheduling policies adjusted to its specificity. Moreover, although the algorithms running in the web service are deterministic (the differences in the processing times of particular attempts for a task execution result from the dynamic character of the environment in which these algorithms are executed not from the randomness of their decisions) we would like to take into account the results obtained in the literature for randomized approaches (cf. e.g. [5]).
References